TUGAS AKHIR – TF 141581

RANCANG BANGUN SISTEM PENGENALAN HURUF DAN ANGKA DALAM SIBI (SISTEM ISYARAT BAHASA INDONESIA) BERBASIS *HAND POSE GESTURE RECOGNITION* MENGGUNAKAN MICROSOFT KINECT

HENDRA IRAWAN
NRP. 2411 100 089

Dosen Pembimbing :
Ir. Apriani Kusumawardhani, M.Sc.

JURUSAN TEKNIK FISIKA
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya
2015

Halaman ini memang dikosongkan

FINAL PROJECT – TF 141581

DESIGN OF LETTERS AND NUMBERS RECOGNITION SYSTEM IN SIBI (SISTEM ISYARAT BAHASA INDONESIA) BASED ON HAND POSE GESTURE RECOGNITION USING MICROSOFT KINECT

HENDRA IRAWAN
NRP. 2411 100 089

Supervisors :
Ir. Apriani Kusumawardhani, M.Sc.

ENGINEERING PHYSICS DEPARTMENT
Faculty of Industrial Technology
Institut Teknologi Sepuluh Nopember
Surabaya
2015

This page intentionally left blank

RANCANG BANGUN SISTEM PENGENALAN HURUF DAN ANGKA DALAM SIBI (SISTEM ISYARAT BAHASA INDONESIA) BERBASIS *HAND POSE GESTURE RECOGNITION* MENGGUNAKAN MICROSOFT KINECT

TUGAS AKHIR
Oleh :

HENDRA IRAWAN
NRP : 2411 100 089

Surabaya, Juli 2015
Mengetahui/Menyetujui,

Pembimbing

Ir. Apriani Kusumawardhani, M.Sc.
NIP. 19530404 197612 2 001

Ketua Jurusan
Teknik Fisika FTI – ITS

Dr. Ir. Totok Soehartanto, DEA
NIP. 19650309 199002 1 001

Halaman ini memang dikosongkan

RANCANG BANGUN SISTEM PENGENALAN HURUF DAN ANGKA DALAM SIBI (SISTEM ISYARAT BAHASA INDONESIA) BERBASIS *HAND POSE GESTURE RECOGNITION* MENGGUNAKAN MICROSOFT KINECT

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Teknik
pada
Bidang Studi Rekayasa Fotonika
Program Studi S-1 Jurusan Teknik Fisika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember

Oleh:
HENDRA IRAWAN
NRP. 2411 100 089

Disetujui oleh Tim Penguji Tugas Akhir :

1. Ir. Apriani Kusumawardhani, M.Sc. ..... (Pembimbing I)
2. Agus M. Hatta ST., M.Si, PhD ..... (Ketua Tim Penguji)
3. Dr.rer.nat.Ir. Aulia M. T. N., M.Sc ..... (Penguji 1)
4. Detak Yan Pratama, ST., M.Sc. ..... (Penguji 2)
5. Dr. Gunawan Nugroho, ST., MT. ..... (Penguji 3)

SURABAYA
JULI 2015

Halaman ini memang dikosongkan

RANCANG BANGUN SISTEM PENGENALAN HURUF DAN ANGKA DALAM SIBI (SISTEM ISYARAT BAHASA INDONESIA) BERBASIS *HAND POSE GESTURE RECOGNITION* MENGGUNAKAN MICROSOFT KINECT

Nama Mahasiswa : Hendra Irawan
NRP : 2411100089
Jurusan : Teknik Fisika FTI-ITS
Dosen Pembimbing: Ir. Apriani Kusumawardhani, M.Sc.

Abstrak

Telah dibangun sistem pengenalan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) berbasis *Hand Pose Gesture Recognition* dengan menggunakan Microsoft Kinect sebagai solusi untuk memudahkan penderita tunarungu dan tunawicara dalam berkomunikasi dengan orang lain disekitarnya. Komponen utama yang digunakan dalam penelitian ini adalah Microsoft Kinect. Tangan-tangan dan jari-jari dideteksi dengan melakukan Segmentasi, *K-means clustering*, *Graham Scan algorithm* dan *Moire contour tracing*, identitas jari-jari dikenali dengan menghitung semua sudut antar jari dan ditentukan nilai ambang batasnya, sedangkan bahasa isyarat dikenali dengan algoritma *Decision Tree Classifier*. Penelitian ini diakhiri dengan karakterisasi sistem pengenalan bahasa isyarat. Sistem pengenalan yang dirancang dan dibangun dapat menerjemahkan huruf dan angka dalam SIBI dengan jarak terjauh yang dapat diterjemahkan sejauh 1.187,6 mm dan jarak terjauh mendeteksi tangan sejauh 1.438 mm dengan kecepatan komputasi sebesar 3,905 ms, dan dengan tingkat akurasi terendah menerjemahkan bahasa isyarat sebesar 30% dan *total success rate* sebesar 83,75%. Sistem juga dapat menerjemahkan bahasa isyarat disetiap kondisi pencahayaan. Medan pandang yang dimiliki oleh sensor Microsoft Kinect sebesar 58 derajat horizontal dan 45 derajat vertikal.

Kata kunci; **SIBI, Microsoft Kinect, *Gesture Recognition*, *Hand Detection***

Halaman ini memang dikosongkan

DESIGN OF LETTERS AND NUMBERS RECOGNITION SYSTEM IN SIBI (INDONESIAN SIGN LANGUAGE SYSTEM) BASED ON HAND POSE GESTURE RECOGNITION USING MICROSOFT KINECT

Student's Name : Hendra Irawan
NRP : 2411100089
Department : Teknik Fisika FTI-ITS
Supervisor : Ir. Apriani Kusumawardhani, M.Sc.

Abstract

Letters and numbers recognition System in SIBI (Sistem Isyarat Bahasa Indonesia) base on Hand Pose Gesture Recognition has been built using Microsoft Kinect as a solution to make it easier for deaf and speech impaired in comunicating with other people around them. The main component used in this research is the Microsoft Kinect sensor. The hands and fingers are detected by doing a segmentation, K –means clustering, Graham Scan algorithm and Moire contour tracing. Identity of the fingers are known by calculating all of the angle between the finger and threshold limit values were specified, while sign language recognized by Decision Tree Classifier algorithm. The research concludes with a characterization of the sign language recognition system. The recognition system designed and built can translate alphabets and numbers in SIBI with the farthest distance which system can be translated is 1,187 .6 mm and the farthest distance to detect hands is 1.438 mm with 3,905 ms computational time, and the lowest accuracy of the system which can translate a sign language is 30% and the total success rate of the system is 83,75%. The system can also translate sign language in every lighting condition. Field of view owned by Microsoft Kinect sensor are 58 degrees horizontally and 45 degrees vertically.

Keywords : SIBI, Microsoft Kinect, Gesture Recognition, Hand Detection

This page intentionally left blank

KATA PENGANTAR

Alhamdulillah, puji syukur senantiasa terpanjatkan kepada Allah SWT yang Maha Segalanya. Atas rahmat, petunjuk dan kasih sayang-Nya, penulis mampu menyelesaikan Tugas Akhir dengan judul :

Rancang Bangun Sistem Pengenalan Huruf dan Angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) Berbasis *Hand Pose Gesture Recognition* Menggunakan Microsoft Kinect

Dalam proses menyelesaikan seluruh pengerjaan tugas akhir ini penulis mendapatkan banyak bantuan dan dukungan dari berbagai pihak. Penulis mengucapkan banyak terima kasih kepada:

1. Bapak Dr.Ir. Totok Soehartanto, DEA selaku Ketua Jurusan Teknik Fisika ITS.
2. Ibu Ir Apriani Kusumawardhani selaku pembimbing yang telah memberi banyak ilmu, pengetahuan, wawasan dan bimbingan moral.
3. Bapak Prof. Dr. Ir. Sekartedjo, M.Sc selaku Kalab Rekayasa Fotonika yang telah memberi banyak ilmu, pengetahuan, wawasan dan bimbingan moral.
4. Totok Ruki Biyanto ST. MT. PhD selaku dosen wali yang telah memberi banyak ilmu, pengetahuan, wawasan dan bimbingan moral
5. Bapak dan Ibu dosen Teknik Fisika yang telah memberi banyak ilmu, pengetahuan, wawasan dan bimbingan moral
6. Ibu, Bapak, dan semua keluarga yang telah memberikan banyak motivasi, doa, dan finansial yang sangat berharga.
7. Angkatan F-46 yang saya banggakan atas motivasi, dukungan dan doanya.

8. Semua mahasiswa tugas akhir bidang Rekayasa Fotonika atas kebersamaan perjuangannya.
9. Semua mahasiswa warga Laboratorium Rekayasa Fotonika atas dukungannya.
10. Semua pihak yang tidak dapat penulis sebutkan satu per satu yang turut membantu dan memperlancar pengerjaan tugas ini. Terima Kasih yang sebesar-besarnya semoga Allah SWT membalasnya dengan pahala yang berlebih. Amin.

Penulis menyadari bahwa tugas akhir ini bukanlah suatu hasil yang sempurna, hanya harapan agar tugas ini menjadi referensi bagi rekan-rekan untuk menambah wawasan bagi pembaca dan dapat digunakan sebagai referensi pengerjaan tugas akhir selanjutnya. Semoga yang sederhana ini dapat menjadi motivasi untuk berkembang lebih hebat lagi.

Surabaya, Juli 2015

Penulis

DAFTAR ISI

DAFTAR TABEL

"Halaman ini sengaja dikosongkan"

DAFTAR SIMBOL

TH_{val}^{i}	Nilai threshold ke-i
P_{min}^{i}	Nilai piksel terendah ke-i
D_B	Nilai kedalaman
TH_{lim}	Nilai threshold batas
D	Jarak antara 2 buah piksel
D_{max}	Jarak terpanjang antara 2 buah piksel
D_{min}	Jarak terpendek antara 2 buah piksel
D_{mean}	Jarak rata-rata antara 2 buah piksel
p	Koordinat suatu titik buah piksel
p_j	Koordinat titik ke-j
μ_i	Nilai rata-rata ke-i
Z_k	Koordinat kedalaman arah z pada titik k
Z_o	Koordinat kedalaman acuan arah z
f	Panjang fokus kamera inframerah
d	Disparitas terukur
b	Lebar ruang medan pandang (mm)
v	Vektor garis
j	Jarak antara 2 buah piksel
θ	Sudut antara 2 buah vektor
X_k	Koordinat objek planimetris arah x pada titik K
Y_k	Koordinat objek planimetris arah y pada titik K
x_k	Koordinat piksel arah x pada titik K
y_k	Koordinat piksel arah x pada titik K
x_o	Koordinat piksel acuan arah x pada titik K
y_o	Koordinat piksel acuan arah x pada titik K
δx	Koreksi ditorsi lensa arah x
δy	Koreksi ditorsi lensa arah y
A	Sudut antara 2 buah vektor (^{o})
$\vec{V}$	Vektor yang dibentuk 2 buah piksel
α	Sudut medan pandang kamera (^{o})

Halaman ini memang dikosongkan

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1. Latar Belakang

Dalam kehidupan sehari-hari selalu ada kebutuhan untuk berkomunikasi menggunakan bahasa isyarat, seperti berkomunikasi dengan tunarungu dan tunawicara. Pada tahun 2009, Badan Pusat Statistik (BPS) menyajikan data statistik disabilitas dalam SUSENAS 2009 dengan kategori kecacatan dengan jumlah total adalah 2.126.998 jiwa di Indonesia, sedangkan penyandang tunarungu mencapai 10,52% dari penduduk Indonesia dengan jumlah 223.738 jiwa dan penyandang tunawicara mencapai 7,12% dari penduduk Indonesia dengan jumlah 151.427 jiwa, hal ini merupakan jumlah yang sangat banyak (Depdiknas, 2009). Kurang dikenalnya bahasa isyarat dalam masyarakat umum membuat penyandang tunarungu dan tunawicara kesulitan dalam berkomunikasi dengan masyarakat sekitar. Selain itu seseorang yang baru saja menyandang tunarungu dan tunawicara juga merasa kesulitan untuk mempelajari bahasa isyarat untuk berkomunikasi. Satu-satunya cara untuk mempelajari bahasa isyarat adalah dengan mendatangi Sekolah Luar Biasa (SLB) yang masih sedikit di Indonesia.

Selain itu, ada beberapa situasi ketika komunikasi tanpa suara lebih dibutuhkan, mislanya selama operasi, ahli bedah harus memberi isyarat kepada perawat untuk meminta bantuan. Sulit bagi kebanyakan orang yang tidak akrab dengan bahasa isyarat untuk berkomunikasi tanpa seorang penerjemah. Dengan demikian, akan sangat membantu apabila terdapat perangkat lunak yang menuliskan simbol dalam bahasa isyarat ke dalam bentuk teks biasa secara

realtime, dan juga berguna untuk memberikan pelatihan interaktif bagi orang-orang dalam belajar bahasa isyarat.

Di Indonesia, bahasa isyarat yang digunakan oleh penderita tunarungu dan tunawicara mengacu pada Sistem Isyarat Bahasa Indonesia atau biasa disebut dengan SIBI. Isyarat pada SIBI secara garis besar terbagi menjadi dua macam, yakni isyarat huruf dan isyarat kata. Untuk isyarat huruf, SIBI berpedoman pada ASL (*American Sign Language*), sedangkan untuk isyarat kata terdapat standar khusus yang telah dibakukan pada kamus Isyarat Bahasa Indonesia (Kamus Sistem Isyarat Bahasa Indonesia, 2001).

Pengenalan Bahasa Isyarat telah menjadi bidang penelitian yang penting dengan fokus pada *Hand Gesture Recognition* (HGR). Penelitian sebelumnya terkait *Hand Gesture Recognition* (HGR) dikenal sebagai *glove-based technologies*. Suatu jenis *glove devices* dirancang sebagai alat input untuk menentukan posisi jari-jari dan telapak tangan di ruang 3D yang digunakan untuk melacak gerakan tangan dan mengenali gerakan. Sebagai contoh, Wii Remote sensor (Wiimote) yang diproduksi oleh Nintendo menyediakan *motion sensing* menggunakan *accelerometer* dan teknologi sensor optik untuk memungkinkan pengguna untuk berinteraksi dengan konsol game. Johnny Lee telah mengembangkan sistem pelacakan jari menggunakan Wii Remote, yang bekerja di ruang 2D dan mampu melacak hingga empat poin jari pada waktu bersamaan. Presisi dan akurasi jarak dari alat ini sangat tinggi. Namun, perangkat ini memiliki kekurangan yaitu bergantung pada beberapa perangkat input yang melekat atau digunakan pada tangan (Lee, 2009).

Secara konvensional, pengenalan gerakan membutuhkan kamera *stereoscopic* dengan kualitas tinggi dan algoritma *computer vision* yang sangat rumit untuk

mengenali isyarat tangan, sistem menjadi mahal dan memerlukan pengaturan yang rumit. *Microsoft Kinect* menyediakan cara murah dan mudah untuk interaksi antara pengguna dan komputer secara *realtime*. Microsoft Kinect, awalnya dirancang untuk game pada platform Microsoft Xbox, menggunakan c*olor sensor* dan *IR depth sensor* untuk menangkap warna gambar (RGB) dan data kedalaman terkait jarak. Hal ini memungkinkan untuk pengembangan algoritma untuk pengenalan data pada gambar. *Driver* perangkat lunak yang dirilis oleh Microsoft yang disebut *Kinect Software Development Kit* (SDK) dengan *Application Programming Interfaces* (API) memberikan akses ke *raw sensor data streams* seperti pendeteksian *skeleton* (rangka tubuh manusia). Namun, tidak ada data pendeteksian tangan yang spesifik yang tersedia untuk pengenalan gerakan, meskipun telah mencakup informasi dari sendi antara tangan dan lengan. Oleh karena itu, pada tugas akhir ini akan dilakukan rancang bangun pengenalan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) berbasis *Hand Pose Gesture Recognition* menggunakan Microsoft Kinect.

1.2. Rumusan Permasalahan

Berdasarkan latar belakang diatas, maka permasalahan yang dapat diambil yaitu, bagaimana merancangan dan membangun sistem pengenalan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) berbasis *Hand Pose Gesture Recognition* menggunakan Microsoft Kinect*?*

1.3. Batasan Masalah

Adapun batas ruang lingkup dari penelitian ini antara lain adalah :

a. Media input yang digunakan sebagai sensor adalah menggunakan Microsoft Kinect untuk Xbox 360.

b. Metode yang digunakan adalah metode *Hand Pose Gesture Recognition*
c. Bahasa isyarat yang dapat dikenali adalah huruf dan angkan dalam SIBI (Sistem Isyarat Bahasa Indonesia).
d. Bagian yang dapat dideteksi adalah bagian jari-jari dan kedua telapak tangan, baik kanan maupun kiri.
e. Dirancang untuk orang Indonesia pada umumnya dengan tinggi 150 – 200 cm.

1.4. Tujuan

Tujuan dari penelitian Tugas Akhir ini antara lain adalah untuk merancang dan membangun sistem pengenalan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) berbasis *Hand Pose Gesture Recognition* menggunakan Microsoft Kinect.

1.5. Manfaat

Manfaat dari tugas akhir ini adalah untuk mendapatkan rancang bangun sistem pengenalan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) berbasis *Hand Pose Gesture Recognition* menggunakan Microsoft Kinect. Penelitian ini mempunyai prospek ke depan yang bermanfaat dalam segi aplikasi. Diharapkan penelitian ini bisa menjadi referensi bagi peneliti lain yang ingin mengembangkan dan menyempurnakan aplikasi dalam bidang pengenalan bahasa isyarat, dan diharapkan penelitian ini terus dilanjutkan dan disempurnakan sehingga bisa diaplikasikan dalam kehidupan nyata.

.

BAB III
METODOLOGI PENELITIAN

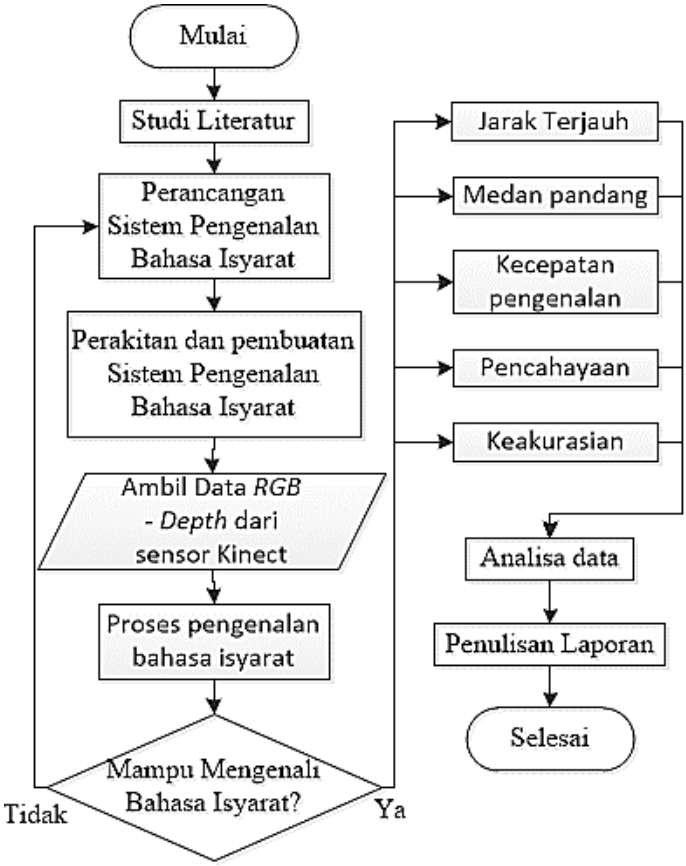
Pada bagian ini dijelaskan mengenai metodologi yang digunakan dalam penelitian ini. Secara umum metode yang dilakukan untuk mencapai tujuan dari tugas akhir ini adalah:

a. Menentukan alat-alat yang akan digunakan, meliputi material dan data yang dibutuhkan
b. Menentukan peralatan yang menunjang terlaksananya penelitian dan perancangan tugas akhir ini, meliputi; alat-alat uji, perangkat keras, perangkat lunak, teori dan persamaan, variabel yang akan digunakan
c. Menentukan proses yang akan digunakan dimulai dari awal penelitian tugas akhir ini digunakan sampai penyimpulan dari akhir proses yang telah dilaksanakan, meliputi: teknik pengumpulan dan analisis data, rancangan, penelitian, cara penafsiran, pengumpulan hasil penelitian, uji coba dan cara evaluasi, dan cara penyimpulan dari keseluruhan proses.

Ketiga langkah tersebut merupakan prosedur penelitian tugas akhir secara umum. Prosedur selengkapnya akan dijelaskan pada gambar 3.1. Penelitian ini dilakukan dengan beberapa tahapan hingga tujuan dapat tercapai. Pengerjaan tugas akhir ini meliputi studi literatur, perancangan dan pembangunan sistem pengenalan SIBI (Sistem Isyarat Bahasa Indonesia), pengujian sistem , analisa data dan penyusunan laporan. Tahapan pengerjaan tugas akhir ini dimulai dengan studi literatur. Studi literatur ini bertujuan untuk mengetahui dasar-dasar perancangan sistem pengenalan bahasa isyarat yang telah dibuat pengembang lain.

Disini sensor yang dipilih sebagai media masukan adalah sensor Microsoft Kinect, sensor ini dipilih karena merupakan sensor yang mudah dikembangkan dalam berbagai *platform* selain itu *library* yang disediakan tersedia secara bebas (*Open Source*) dan yang paling utama adalah Microsoft Kinect menyediakan sensor RGB dan kedalaman dalam satu perangkat keras. Desain tersebut kemudian akan digunakan ketika perancangan. Setelah

desain selesai dibuat, maka dilakukan perakitan dan pembuatan sistem pengenalan bahasa isyarat. Setelah selesa perakitan dan pembuatan, maka dilakukan proses pengenalan SIBI sehingga mampu diterjemahkan kedalam bentuk teks. Jika ada kegagalan dalam penerjemahan bahasa isyarat, maka dilakukan perbaikan.



Gambar 3.1 Diagram Alir Penelitian

Kemudian sistem ini akan di uji untuk mendapatkan spesifikasi dari sistem itu sendiri mencakup jarak terjauh, medan pandang, kecepatan komputasi sistem pengenalan, pengaruh

tingkat pencahayaan dan keakurasian. Setelah didapatkan data tersebut, maka akan dilakukan analisa. Terakhir dilakukan penulisan laporan.

3.1 Peralatan dan Bahan

Peralatan dan bahan yang digunakan dalam tugas akhir ini adalah:

- Meteran

Gambar 3.2 Meteran

- Busur

Gambar 3.3 Busur

- Lux Meter

Gambar 3.4 Lux Meter

- Laptop / PC

Gambar 3.5 Laptop / PC

- Microsoft Kinect untuk XBOX 360

Gambar 3.6 Microsoft Kinect untuk Xbox 360

- *Display monitor*

Gambar 3.7 *Display Monitor*

3.2 Perancangan Sistem Pengenalan Bahasa Isyarat

Perancangan sistem pengenalan bahasa isyarat dimulai dari desain sistem pengenalan yang meliputi :

- Algoritma *Hand Detection*
- Algoritma *Finger Identification*
- Algoritma *Gesture Recognition*

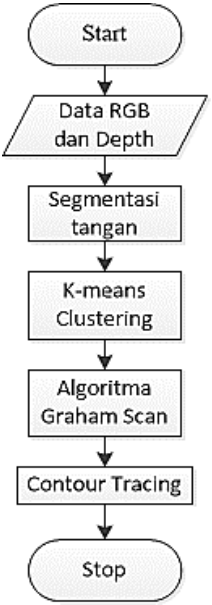
Sedangkan untuk media masukan berupa sensor Microsoft Kinect untuk XBOX 360 tidak perlu lagi dilakukan perancangan apapun karena akan digunakan tanpa ada modifikasi.

3.2.1 Algoritma *Hand Detection*

Hand Detection adalah pemisahan piksel-piksel tangan dengan latarbelakang dan diproses sehingga dikenali mana tangan kanan dan kiri. Awalnya dilakukan segmentasi untuk memisahkan tangan dengan latarbelakang sehingga selanjutnya bisa dilakukan pengenalan tangan. Dalam sistem pendeteksian tangan, sistem ini berjalan secara *realtime* dan *K-means clustering* diterapkan setiap kali ada perubahan dalam sumber data masukan.

Setelah piksel-piksel tangan terkumpul, maka dilakukan deteksi *convex hull* dan kontur tangan. *Convex hull* dicari dengan menggunakan algoritma *Graham Scan* dan kontur tangan dicari menggunakan algoritma modifikasi *Moore-Neighbor tracing*. Setelah dilakukan pengelompokan, sekelompok piksel-piksel ditemukan dan disimpan. Setelah kontur tangan terdeteksi, pusat

telapak tangan kemudian dihitung. Posisi dari pusat telapak tangan (*palm centre*) digunakan untuk menghitung arah dan sudut dari jari-jari tangan.



Gambar 3.8 Diagram Alir Algoritma *Hand Detection*

Segmentasi

Microsoft Kinect sebagai sensor kedalaman memiliki rentang operasi 800 - 3500 mm, menghasilkan *depth image* dengan resolusi VGA (640 x 480), dan mampu menangkap *depth image* hingga 30 fps. Setiap piksel pada *depth image* diukur dalam skala milimeter. *Depth image* perlu diubah menjadi gambar grayscale untuk memvisualisasikan data sebagai gambar terlihat.

Proses Segmentasi tangan digambarkan pada gambar 3.9. Untuk mendeteksi tangan, langkah pertama adalah memisahkan tangan dari latar belakang. Nilai ambang batas data kedalaman diatur secara manual untuk menentukan kisaran kedalaman dimana gerakan tangan akan dikenali. Segmentasi hanya bekerja dengan beberapa asumsi yaitu : a) tubuh manusia dalam *depth image* adalah objek paling dekat dengan sensor kedalaman, b)

tangan manusia diposisikan di depan tubuh, dan c) tidak ada benda di sekitar tubuh manusia yang menghalanginya.

Gambar 3.9 Proses Segmentasi Tangan

Segmentasi tangan dilakukan dengan menerapkan operasi *image threshold* ke dalam *image depth*. Operasi batas ini hanya menghilangkan latarbelakang dan menampilkan tangan. Untuk menentukan *threshold* (TH_{hand}), objek dengan jarak terdekat pada *depth image* harus dicari dengan mencari nilai piksel terendah dalam gambar (P_{min}). Mengacu pada asumsi bahwa tangan manusia selalu diposisikan di depan tubuh, lokasi kedua tangan dapat ditemukan dengan mencari objek jarak terdekat. Karena jarak lebih dekat maka semakin rendah intensitasnya, dengan demikian nilai piksel terendah di masing-masing daerah ROI (*Region Of Interest*) dicari dengan persamaan sama seperti pada segmentasi tubuh. Ambang batas dapat dihitung dengan menggunakan :

$$TH_{val}^{i} = P_{min}^{i} + \left(255 \times \left(\frac{D_B - 800}{3500 - 800}\right)\right) \qquad (3.1)$$

dimana D_B adalah kedalaman yang diperoleh dari sensor. Untuk menghindari kesalahan segmentasi yang disebabkan oleh sebagian dari bagian tubuh lain yang tersegmentasi, nilai TH_{val}

masing-masing daerah dievaluasi dengan menggunakan persamaan :

$$TH_{val}^{i} = \min\left(TH_{val}^{i}, TH_{lim}\right) \quad (3.2)$$

dimana TH_{lim} adalah batas nilai ambang yang telah ditentukan yaitu 1000 – 1200 mm. Selanjutnya, piksel tangan ini diproyeksikan ke ruang 2D untuk analisa selanjutnya. Jarak antara dua piksel $p_1(x_1, y_1)$ dan $p_1(x_2, y_2)$ didefinisikan sebagai:

$$D(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (3.3)$$

Clustering

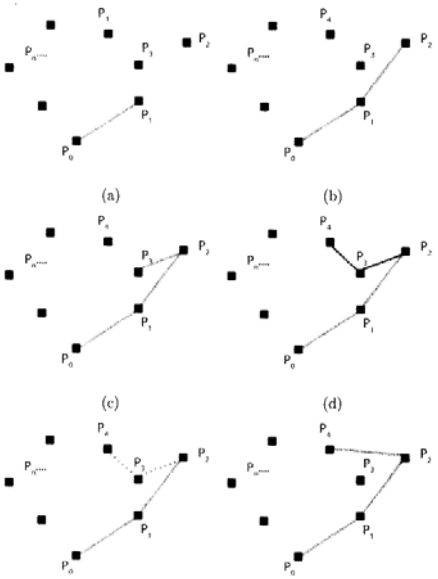
Selanjutnya, piksel tangan ini diproyeksikan ke ruang 2D, algoritma *K-means clustering* diterapkan untuk membagi semua piksel ke dalam dua kelompok yaitu kelompok tangan kanan dan kiri. *K-means clustering* terus dilakukan setiap kali ada perubahan dalam sumber input data. Pada awal setiap iterasi, setiap kelompok kosong diinisialisasi dengan titik-titik acak yang terletak dalam rentang nilai *treshold* yang telah ditentukan sebagai nilai rata-rata. Setelah *K-mean* konvergen, titik-titik dari masing-masing tangan dikelompokkan. Jika jarak antara dua *centroid* tangan kurang dari nilai yang telah ditentukan, maka kedua kelompok akan digabung menjadi satu.

Menemukan *convex hull* dan *contour tracing*

Setelah titik-titik telah terkelompok maka dilakukan deteksi *convex hull* serta kontur dari tangan. Algoritma *Graham Scan* digunakan untuk menghitung *convex hull* dari kelompok tangan yang terdeteksi.

Graham Scan Algorithm : Algoritma ini termasuk algoritma runut balik, karena jika solusi yang coba dibentuk tidak sesuai maka akan dilakukan runut balik untuk memilih titik berikutnya. proses pencarian *convex hull* oleh algoritma *Graham Scan* digambarkan pada gambar 3.10. Garis besar algoritma ini terdiri dari 3 tahap, yaitu :

a. Memilih titik luar yang akan menjadi poros. Titik luar adalah titik unik yang memiliki kriteria tertentu. Dalam penelitian ini, titik luar didefinisikan sebagai titik yang memiliki nilai sumbu *Y* terbesar. Jika terdapat 2 nilai yang sama, maka dipilih titik dengan nilai sumbu *X* terkecil
b. Melakukan pengurutan terhadap titik-titik sisa secara mengecil berdasarkan besar sudut relatif terhadap titik luar
c. Gambar garis lintang terhadap titik yang sudah terurut. Bentuk sisi baru jika sudut yang dibentuk lebih kecil dari 180° atau lakukan runut balik jika lebih besar dari 180°.



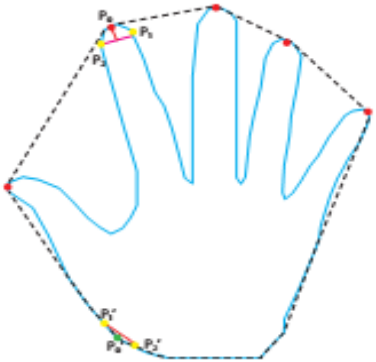
Gambar 3.10 Algoritma *Graham Scan* untuk Mendeteksi *Convex Hull* (Graham, 1972)

Sejalan dengan pendeteksian *convex hull*, deteksi kontur tangan juga dilakukan, kontur tangan terdeteksi oleh modifikasi algoritma *Moore-Neighbor tracing*. Setelah pengelompokan, sekelompok titik-titik piksel tangan yang dideteksi lalu disimpan. Setelah kontur tangan terdeteksi, pusat dari telapak tangan

dihitung sebagai pusat *inscribing circle* dari kontur tangan. Posisi pusat telapak tangan digunakan untuk menghitung arah jari.

3.2.2 Algoritma *Finger Identification*

Untuk mengenali bahasa isyarat maka diperlukan pengenalan jari-jari apa saja yang terbentang pada tangan. Digunakan *library* Candescent NUI untuk mendeteksi jari-jari pada tangan baik kanan maupun kiri. Algoritma pendeteksian jari merupakan *multi-thread algorithm*, yang memiliki dua *thread* yang terpisah yaitu satu untuk tangan kanan dan satu untuk tangan kiri. Hal ini memungkinkan pendeteksian jari-jari bekerja secara paralel agar pendeteksian dan pelacakan jari bekerja secara optimal. Algoritma pengenalan jari dilakukan sebagai berikut :



Gambar 3.11 Pendeteksian Ujung Jari

Langkah 1 : pendeteksian ujung jari (*fingertip*)

Ujung jari terdeteksi dengan memeriksa hubungan keselarasan tiga titik. Titik-titik yang diterapkan algoritma hubungan keselarasan tiga titik adalah titik-titik yang berada pada *convex hull* dan kontur tangan. untuk mempercepat perhitungan setiap titip diperiksa dengan algoritma seperti yang digambarkan pada gambar 3.11, yaitu :

a) Misalkan C adalah himpunan semua titik-titik yang ada pada *convex hull* dan *hand contour*

b) Untuk setiap titik p_0 di C, ambil 2 titik $p_1$dan p_2 di dua arah yang berlawanan sepanjang kontur yang berada pada jarak tertentu untuk p_0
c) Temukan titik tengah $p_1$dan p_2 dan hitung jarak antara titik tengah ini dan p_0. Jika jaraknya lebih besar dari nilai tertentu, maka tiga poin tidak sejajar. Sehingga kemudian p_0 ini diidentifikasi sebagai ujung jari. Jika tidak, kembali ke Langkah 2 dan memeriksa kembali titik-titik pada *convex hull* dan *hand contour* selanjutnya.

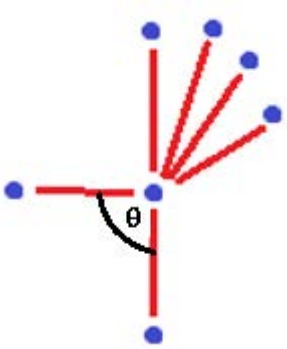
Metode ini digambarkan pada gambar 3.11. Titik-titik merah menggambarkan ujung jari yang benar, dan titik hijau bukanlah ujung jari. Titik-titik kuning digunakan untuk memeriksa keselarasan tiga titik. Jarak antara titik p_0 dan garis yang dibuat oleh titik-titik p_1 dan p_2 ternyata lebih besar dari titik p'_0 dan garis yang dibuat oleh titik-titik p'_1 dan p'_2. Sehingga jarak *threshold* dapat digunakan untuk membedakan titik ujung jari dan titik bukan ujung jari. Setelah ujung jari terdeteksi, vektor arah dari jari-jari akan mudah untuk dihitung dengan mengurangi posisi pusat telapak tangan $P_c(x_c, y_c)$. Vektor-vektor ini terproyeksi dari pusat telapak tangan ke ujung jari.

Langkah 2 : *buffer data* dan *noise filter*

Selanjutnya, titik-titik ujung jari yang terdeteksi dibuffer. Titik-titik tersebut harus terdeteksi dalam waktu tertentu sebelum dilakukan pelacakan terhadap titik-titk tersebut. Setiap titik yang berkedip dianggap sebagai *noise* dan diabaikan. Setelah titik dilacak cukup lama dan dianggap bukan *noise*, maka ditambahkan ke memori untuk disimpan dan diproses.

Langkah 3 : penggambar vektor-vektor jari

Berikutnya, dibangun sebuah vektor yang mewakili pergelangan tangan, dengan mengurangi koordinat titik pergelangan tangan itu dari koordinat titik tangan. Untuk setiap titik jari yang akan diproses, dihitung juga vektor dari setiap jari dengan mengurangi koordinat ujung jari dengan koordinat telapak tangan. seperti pada gambar 3.12.

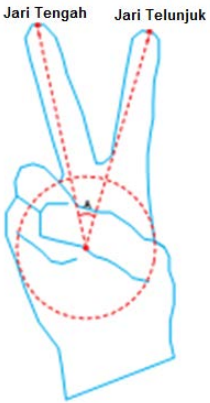


Gambar 3.12 Vektor Jari-jari Terhadap Telapak Tangan

Langkah 4 : perhitungan sudut dan resultan vektor jari-jari

Lalu ukur semua jarak antar jari-jari yang terbuka, resultan vektor jari-jari, dan sudut dari telapak tangan dengan jari-jari yang terbuka dihitung searah jarum jam untuk tangan kanan dan berlawanan arah jarum jam untuk tangan kiri. Sudut A antara dua buah vektor $\overrightarrow{V_1}(x_1, y_1)$ dan $\overrightarrow{V_2}(x_2, y_2)$ dihitung sebagai:

$$A = \arccos \frac{\overrightarrow{V_1}.\overrightarrow{V_2}}{\|\overrightarrow{V_1}\|\|\overrightarrow{V_2}\|} \tag{3.4}$$



Gambar 3.13 Skema Perhitungan Sudut antara 2 Buah Vektor Jari

Setelah semua sudut dihitung, semua jari diurutkan dari yang terkecil hingga terbesar berdasarkan sudut mereka, kemudian diukur perbedaan relatif antara masing-masing sudut jari dengan pasangan tetangganya untuk menentukan identitas jari mereka. Tabel 3.1 dan tabel 3.2 menunjukkan nilai-nilai ambang batas dalam penentuan pengenalan jari-jari.

Tabel 3.1 Nilai-nilai Ambang Batas Pengenalan 1 Jari

Identitas Jari	Sudut Ambang
Thumb	< 130
Index	130 -184
Middle	185 - 214
Ring	215 - 225
Pinkie	> 225

Tabel 3.2 Nilai-nilai Ambang Batas Pengenalan antara 2 Jari

Jari 1	Jari 2	Sudut Ambang
Thumb	Index	73-110
Thumb	Middle	99-121
Thumb	Pinkie	118-180
Index	Middle	22-43
Index	Pinkie	55-90
Middle	Ring	22-45
Middle	Pinkie	51-70
Ring	Pinkie	28-50

Ada beberapa tumpang tindih pada sudut ambang, tetapi sudut ambang memiliki nilai yang cukup untuk dapat menentukan ambang batas di antara jari -jari.

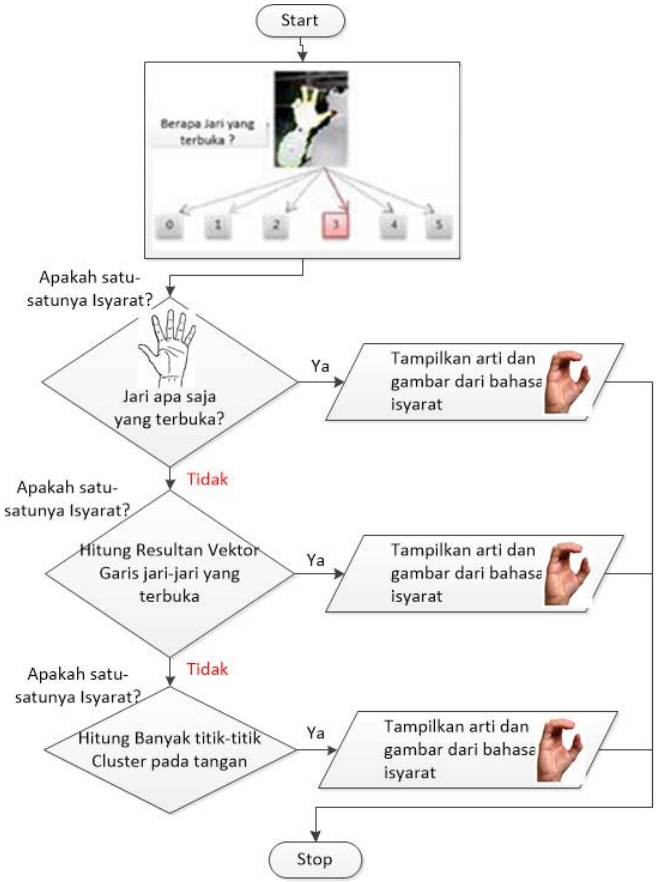
Langkah 5 : pengenalan sisa jari-jari

Kemudian jari-jari yang tak bisa dikenali, dikenali dengan digram pohon sebagai berikut :

- Jika ada satu titik, maka jari ditentukan dari sudut antara pergelangan tangan dan titik jari tersebut
- Jika ada lebih dari satu titik, maka dibandingkan sudut antara dua jari yang pertama
  - Jika kurang dari 60 derajat, titik pertama adalah jari telunjuk dan titik kedua adalah jari tengah
  - Jika kurang dari 100 derajat, titik pertama adalah ibu jari dan kedua adalah jari telunjuk
  - Jika kurang dari 120 derajat, titik pertama adalah ibu jari dan yang kedua adalah jari tengah
  - Jika sama dengan atau lebih dari 120 derajat, titk pertama adalah ibu jari dan yang kedua adalah jari kelingking
- Jika ada titik jari yang tersisa, maka dibandingkan sudut jari dengan jari sebelahnya dan dilakukan sampai titiik jari yang terakhir dikenali sebagai jari kelingking
  - Jika yang terakhir adalah jari telunjuk dan sudut kurang dari 50 derajat, maka dikenali sebagai jari tengah
  - Jika yang terakhir adalah jari telunjuk dan sudut sama dengan atau lebih dari 50 derajat, maka dikenali sebagai jari kelingking
  - Jika yang terakhir adalah jari tengah dan sudut kurang dari 45 derajat, maka dikenali sebagai jari manis
  - Jika yang terakhir adalah jari tengah dan sudut sama dengan atau lebih dari 45 derajat, maka dikenali sebagai jari kelingking
  - Jika yang terakhir adalah jari manis, maka dikenali sebagai jari kelingking.

3.2.3 Algoritma *Gesture Recognition*

Setelah dilakukan *Finger Identification*, maka dilakukan *Gestures Recognition* yang melewati empat lapis pengklasifikasian, yaitu : *Finger Counting Classifier*, *Finger Name Colecting*, *Vector Matching* dan *Cluster Point Counting*. Untuk mengenali suatu bahasa isyarat maka dilakukan algoritma untuk membedakan bahasa isyarat yang satu dan yang lainnya dengan menggunakan algoritma *Decision Tree Classifier,*



Gambar 3.14 Algoritma *Decision Tree Classifier* untuk Pengenalan Bahasa Isyarat

***Decission Tree Classifiers* :**

a) *Finger counting classifier* : Gerakan pertama kali diklasifikasikan dengan jumlah jari yang diacungkan, dan kemudian dikirim ke layer kedua dari pengklasifikasi.
b) *Finger name colecting* : Gerakan diklasifikasikan lebih lanjut dengan mengidentifikasi nama jari-jari yang diacungkan.
c) *Vector matching* : Arah vektor dari semua jari yang diacungkan diukur, dan semua sudut yang terdapat antara jari-

jari yang diacungkan dihitung. Pada layer ini arti dari gerakan ini diklasifikasikan menurut sudut tersebut.

d) *Cluster Point Counting* : jumlah titik-titik hasil clustering pada setiap tangan dihitung, pada layer ini arti dari bahas isyarat diklasifikasikan menurut luasan permukaan dari tangan yang dideteksi.

Dalam mengenali suatu bahasa isyarat dilakukan Algoritma seperti pada gambar 3.14. Data-data yang telah didapat sebelumnya digunakan dalam algoritma *Decision Tree Classifier*. Pada algoritma ini, tangan yang telah terdeteksi sebelumnya, kemudian dihitung titik jari-jari yang terdeteksi, setelah itu dilakukan pengenalan titik-titik jari yang terdeteksi dengan algoritma *Finger Identification* dan didapatkan identitas jari-jari maka pada *layer* ini dilakukan pencocokan pada *database*, ketika tidak ada gerakan yang sejenis pada *database* maka bahasa isyarat dikenali dan ditampilkan gambar serta arti dari bahasa isyarat tersebut, namun jika ada yang sejenis maka data-data dikirimkan ke *layer* selanjutnya yaitu *Vector Matching*.

Pada *Vector Matching* dilakukan perhitungan resultan-resultan dan arah dari vektor-vektor jari yang terdeteksi, setelah didapat besar dan arah dari resultan jari-jari yang terbuka maka dilakukan pencocokan pada *database*, ketika tidak ada gerakan yang sejenis pada *database* maka bahasa isyarat dikenali dan ditampilkan gambar serta arti dari bahasa isyarat tersebut, namun jika tidak ada yang sejenis maka data-data dikirimkan ke layer selanjutnya yaitu *Cluster Point Counting.*

Pada layer ini dihitung jumlah *cluster points* yang ada pada tangan, *cluster points* ini adalah hasil dari proses *K-Means Clustering* yang telah dijelaskan pada subbab sebelumnya, setelah didapatkan jumlah *cluster points* maka dilakukan pencocokan pada *database* bahasa isyarat, bila terdapat kecocokan pada *database* bahasa isyarat maka bahasa isyarat dikenali dan ditampilkan gambar dan arti dari bahasa isyarat tersebut. Namun, karena layer ini merupakan layer yang terakhir, maka bila terdapat kesamaan jumlah pada *cluster points*, bahasa isyarat tersebut tidak dapat dikenali.

Tabel 3.3 Tabel Pengenalan Huruf dalam SIBI

Bahasa Isyarat	**Jari yang Terbuka**	**Identitas Jari yang Terbuka**	**Arah Vektor Jari (derajat)**	**Jumlah *Cluster Point* (piksel)**
A	0	-	-	140-155
B	4	*index-middle-ring-pinkie*	-	-
C	1	*thumb*	(-180) - (-165)	190-230
D	1	*index*	90-110	105-125
E	0	-	-	155-190
F	3	*middle-ring-pinkie*	-	-
G	1	*thumb*	155-170	120-165
H	2	*index-middle*	177-155	140-190
I	1	*pinkie*	-	-
K	2	*thumb-index*	105-125	130-170
L	2	*thumb-index*	125-140	-
O	0	-	-	125-140
P	2	*thumb-index*	(-180)-(-160)	130-156
Q	2	*thumb-pinkie*	(-110)-(-80)	-
R	1	*index-middle*	90-110	120-150
S	0	-	-	110-125
T	0	-	-	90-110
U	1	*index*	85-110	130-160
V	2	*index-middle*	75-115	-
W	3	*index-middle-ring*	-	-
X	1	*index*	110-130	-
Y	2	*thumb-pinkie*	90-110	-

Tabel 3.4 Tabel Pengenalan Angka dalam SIBI

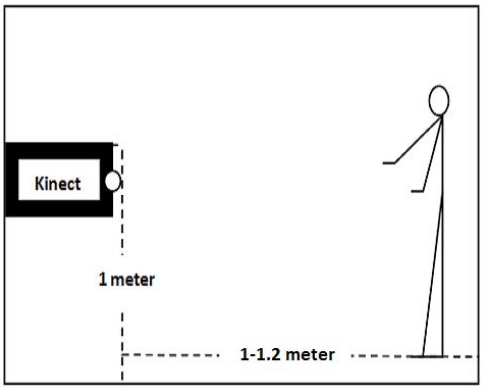
Bahasa Isyarat	**Jari yang Terbuka**	**Identitas Jari yang Terbuka**
0	0	-
1	1	*index*
2	2	*index-middle*
3	3	*thumb-index-middle*
4	4	*index-middle-ring-pinkie*
5	5	*thumb-index-middle-ring-pinkie*
6	3	*index-middle-ring*
7	3	*index-middle-pinkie*
8	3	*index-ring-pinkie*
9	3	*middle-ring-pinkie*

3.3 Pembuatan Sistem Pengenalan Bahasa Isyarat

Setelah perancangan sistem bahasa isyarat dilakukan, maka langkah selanjutnya adalah merakit dan melakukan pengaturan sistem pengenalan bahasa isyarat. perancangan pengaturan sistem pengenalan ini dilakukan untuk mendapatkan hasil terbaik dalam mendeteksi tangan-tangan, jari-jari tangan baik kanan maupun kiri, sehingga mampu mengenali bahasa isyarat dengan memenuhi kriteria-kriteria yang diharapkan. Untuk mencari

pengaturan yang baik dan memenuhi kriteria yang diharapkan maka dilakukan langkah-langkah sebagai berikut :

a) Dilakukan variasi jarak antara sensor Microsoft Kinect dengan tangan dari jarak 0 cm dengan kenaikan 10 cm
b) Dillihat apakah tedeteksi kedua tangan dan dikenali setiap jari-jari yang ada pada tangan kanan maupun kiri
c) Jika tidak tedeteksi kedua tangan dan dikenali setiap jari-jari, maka ulangi langkah a, sampai batas akhir jarak tangan dan jari-jari terdeteksi dan dikenali.



Gambar 3.15 Pengaturan Sistem Pengenalan Bahasa Isyarat

Pengaturan penelitian dirancang sehingga semua jari dan telapak tangan dapat dikenali dengan baik dan dengan sedikit *noise* yang ditangkap, dan didapatkan bahwa jarak antara sensor dan tangan adalah 1.000-1.200 mm karena hasil *Hand Detection* didapatkan hasil yang terbaik dan *depth thresholding* diatur dalam rentang 800–1.200 mm. Tinggi peletakan sensor Microsoft Kinect ditetapkan 1000 mm karena disesuaikan dengan tinggi rata-rata orang Indonesia pada umumnya.

Untuk software yang digunakan dalam sistem pengenalan bahasa isyarat ini, digunakan bahasa C# untuk menuliskan semua logika pengenalan bahasa isyarat. Sedangkan IDE (*Integrated Development Environment*) yang digunakan adalah menggunakan Visual Studio 2013 *student version*.

3.4 Pengujian

Untuk mendapatkan spesifikasi dari sistem pengenalan bahasa isyarat maka dilakukan pengujian yang meliputi :

a. Jarak terjauh dimana sistem pengenalan dapat mengenali kelima jari
b. Medan pandang dari Microsoft Kinect (*Field of View*)
c. Kecepatan komputasi dari sistem pengenalan bahasa isyarat
d. Pengaruh perubahan cahaya terhadap penerjemahan bahasa isyarat
e. Keakurasian dalam menerjemahkan bahasa isyarat.

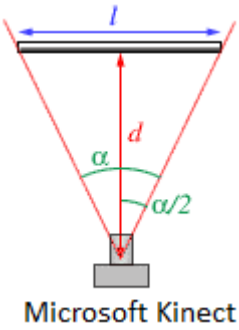
3.4.1 Jarak Terjauh

Pengujian ini di tujukan untuk menguji spesifikasi sistem pengenalan bahasa isyarat untuk menerjemahkan bahasa isyarat pada jarak terjauh yang masih dapat dideteksi oleh sensor Microsoft Kinect. Pengujian dilakukan sebanyak 10 kali dengan 5 kali tangan kanan dan 5 kali tangan kiri pada pengujian pengenalan kelima jari maupun pengenalan tangan saja.

3.4.2 Medan Pandang (*Field of View*)

Pengujian ini dilakukan untuk mendapatkan jangkauan yang dapat dideteksi sensor, baik jangkauan kiri dan kanan maupun atas dan bawah. Berdasarkan spesifikas sensor Microsoft Kinect untuk XBOX 360 yang diterbitkan oleh Microsoft, diketahui bahwa medan pandang (*field of view*) horizontal adalah 58 derajat, sedangkan medan pandang (*field of view*) vertikal adalah 45 derajat. Untuk memvalidasinya dilakukan pengambilan gambar dari sensor Microsoft Kinect dengan sudut jangkauan maksimal dan diukur menggunakan meteran dan busur kemudian dilakukan perhitungan sehingga didapatkan medan pandang dari sensor Microsoft Kinect. Skema Perhitungan medan pandang (*Field of View*) pada Microsoft Kinect ditunjukkan pada gambar 3.16 dengan persamaan beikut ini :

$$\alpha = 2 \times \tan^{-1}\left(\frac{l}{2d}\right) \qquad (3.5)$$



Gambar 3.16 Skema Perhitungan *Field of View* dari Sensor Microsoft Kinect

3.4.3 Kecepatan Komputasi

Pengujian kecepatan komputasi dilakukan untuk mendapatkan rentang kecepatan sistem dalam menerjemahkan SIBI (Sistem Isyarat Bahasa Indonesia) dalam satu kali pernerjemahan bahasa isyarat. Sehingga dapat diketahui apakah sistem cocok untuk diterapkan. Pengujian dilakukan dalam menerjemahkan angka 0-9 dalam SIBI. Sehingga total didapatkan 10 data kecepatan komputasi.

3.4.4 Pengaruh Pencahayaan

Pengujian ini ditujukan untuk menguji spesifikasi sistem pengenalan bahasa isyarat untuk melihat pengaruh kuat penerangan pada penerjemahan bahasa isyarat. Besar kuat penerangan yang di ukur adalah sebesar 198, 74, 32, 17, 05 dan 0 (Gelap) lux. Setiap tingkat cahaya di ambil 10 kali penerjemahan untuk menerjemahkan angka 0-9 dalam SIBI sehingga total didapat 60 data.

3.4.5 Akurasi Pengenalan Bahasa Isyarat

Pengujian ini bertujuan untuk menguji keakurasian sistem pengenalan bahasa isyarat dalam menerjemahkan huruf dan angka dalam SIBI (Sistem Isyarat Bahas Indonesia). Bahasa isyarat yang diujikan adalah bahasa isyarat angka dan huruf dalam SIBI yaitu angka 0-9 dan A-Z kecuali huruf J, Z, M, dan N. Kondisi

pengujian dilakukan pada kondisi ideal yaitu dengan kondisi kuat penerangan 198 lux dan jarak antara sensor Microsoft Kinect dan tangan sejauh 1100 mm. Pengujian dilakukan sebanyak 10 kali setiap bahasa Isyarat dengan 5 kali menggunakan tangan kiri dan 5 kali dengan tangan kanan, sehingga total data yang dihasilkan sebanyak 320 data.

3.5 Analisis Data dan Pembahasan

Analisis data dilakukan terhadap hasil-hasil pengukuran yang diperoleh. Dari hasil tersebut, akan dicari spesifikasi pada setiap parameter-parameter pengujian yang telah dijelaskan sebelumnya sehingga didapatkan spesifikasi dan kemampuan sistem pengenalan bahasa isyarat untuk menerjemahkan suatu bahasa isyarat.

3.6 Penulisan Laporan

Tahap akhir dari penelitan ini adalah penyusunan laporan. Laporan tersebut merupakan rekaman dan bentuk pertanggung jawaban dari berbagai kegiatan yang dilakukan selama proses penelitian kepada pihak terkait. Setelah itu, hasilnya akan disampaikan baik secara tulisan dan lisan.

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dijelaskan mengenai beberapa teori penunjang dalam perancangan bangun pengenalan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) berbasis *Hand Pose Gesture Recognition* menggunakan Microsoft Kinect.

2.1 Bahasa Isyarat

Bahasa isyarat adalah bahasa yang mengutamakan komunikasi visual, bahasa tubuh, dan gerak bibir untuk berkomunikasi dan bukan dengan suara. Penderita tunarungu dan tunawicara adalah kelompok utama yang menggunakan bahasa ini, biasanya dengan mengkombinasikan bentuk tangan, orientasi dan gerak tangan, lengan, dan tubuh, serta ekspresi wajah untuk mengungkapkan pikiran mereka.

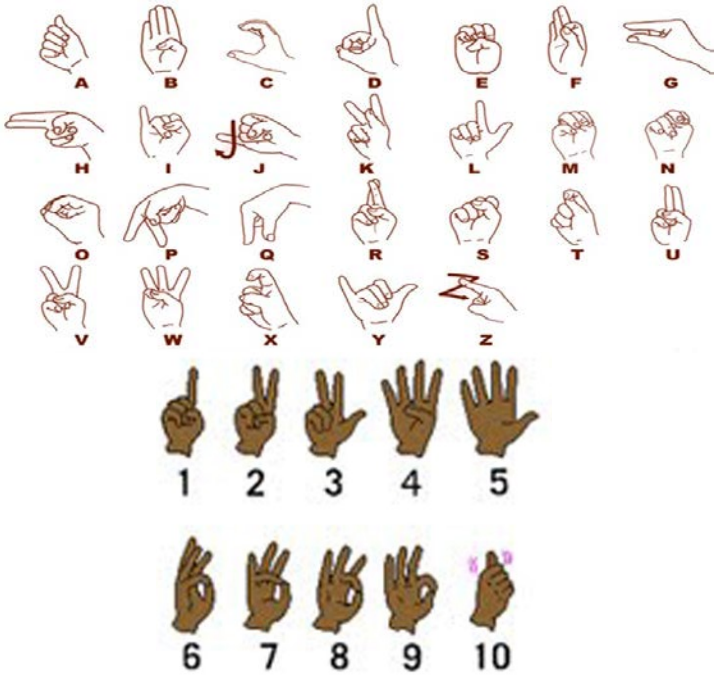
Bahasa isyarat pada dasarnya digunakan oleh orang-orang untuk mengekspresikan makna dari apa yang diucapkan. Setiap pengguna bahasa isyarat harus juga mengekspresikan bahasa tubuh mereka dan maksud yang ingin dicapai.

Untuk Indonesia, sistem yang sekarang umum digunakan ada dua sistem yaitu BISINDO (Bahasa Isyarat Indonesia) yang dikembangkan oleh tunarungu sendiri melalui GERKATIN (Gerakan Kesejahteraan Tunarungu Indonesia) dan SIBI (Sistem Isyarat Bahasa Indonesia).

2.1.1 SIBI (Sistem Isyarat Bahasa Indonesia)

SIBI (Sistem Isyarat Bahasa Indonesia) merupakan salah satu media yang membantu komunikasi sesama kaum tunarungu dan tunawicara ataupun komunikasi kaum tunarungu dan tunawicara dengan orang-orang normal disekitarnya yang disusun oleh pemerintah Indonesia sebagai bahasa isyarat resmi di Indonesia. SIBI kebanyakan mengadopsi kepada *Amercan Sign Language* (ASL). Wujudnya adalah tatanan yang sistematis bagi seperangkat isyarat jari, tangan, dan berbagai gerak untuk

melambangkan kosa kata Bahasa Indonesia (Kamus Sistem Isyarat Bahasa Indonesia, 2001).



Gambar 2.1 Contoh SIBI (Sistem Isyarat Bahasa Indonesia) (Kamus Sistem Isyarat Bahasa Indonesia, 2001)

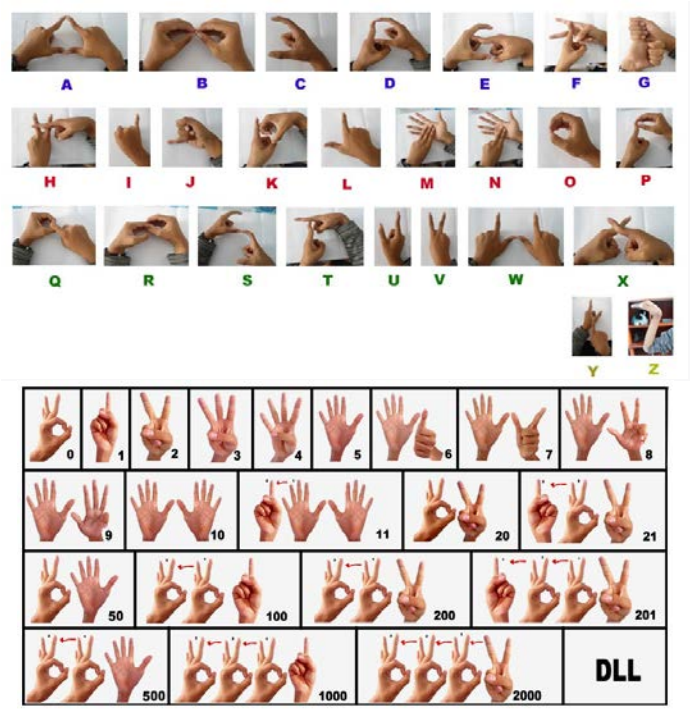
Selain isyarat kata, dalam sistem ini tercakup pula sistem ejaan jari yang digunakan untuk mengisyaratkan :

- Nama diri
- Singkatan atau akronim
- Bilangan
- Kata yang belum memiliki isyarat,

selanjutnya dalam berkomunikasi dengan sistem ini tidak berbeda dengan cara komunikasi secara lisan, yaitu aturan yang berlaku pada bahasa lisan berlaku pula pada sistem isyarat ini, hanya saja intonasi dilambangkan dengan mimik muka, gerak bagian tubuh, kelenturan, dan kecepatan dalam berisyarat (Kamus Sistem Isyarat Bahasa Indonesia, 2001).

2.1.2 BISINDO (Bahasa Isyarat Indonesia)

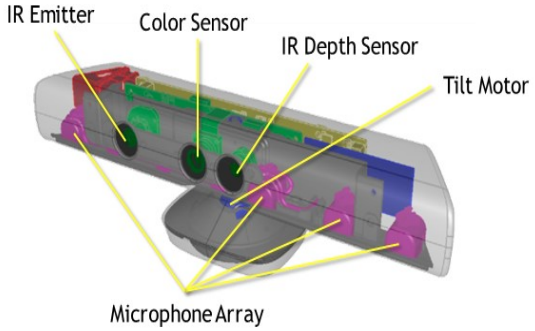
BISINDO (Bahasa Isyarat Indonesia) adalah bahasa ibu komunitas tunarungu di Indonesia. BISINDO berkembang secara alami di kalangan tunarungu Indonesia dan sudah digunakan turun temurun selama bertahun-tahun. Walaupun sudah menjadi bahasa pengantar sejak lama, tetapi keberadaan BISINDO secara resmi belum diakui oleh pemerintah. BISINDO tidak digunakan di sekolah-sekolah luar biasa dan mayoritas media massa belum menggunakan BISINDO. Saat ini pemerintah hanya mengakui SIBI (Sistem Isyarat Bahasa Indonesia) sebagai satu-satunya bahasa isyarat resmi. BISINDO digunakan untuk berkomunikasi antar individu sebagaimana sama seperti halnya dengan bahasa Indonesia pada umumnya (Gerkatin, 2014).



Gambar 2.2 Contoh BISINDO (Bahasa Isyarat Indonesia) (Gerkatin, 2014)

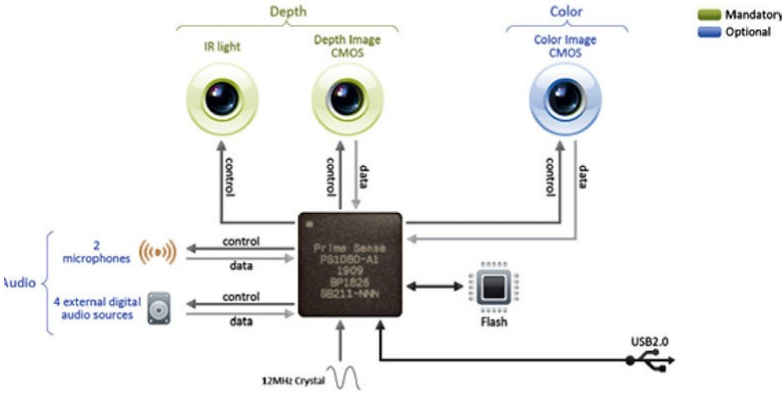
2.2 Microsoft Kinect

Microsoft Kinect adalah sebuah aksesori untuk *platform* game konsol Microsoft Xbox 360. Kinect merupakan sebuah *motion sensing unit device* yang dikembangkan untuk menginterpretasikan gerak tubuh manusia. Microsoft Kinect memiliki sensor-sensor yang terhubung dengan *motorized pivot based*. Perangkat ini memiliki kamera RGB, *depth sensor*, dan *multi-array microphone*. Dengan sensor-sensor tersebut Microsoft Kinect dapat menyajikan kemampuan melakukan *full-body 3D motion capture, facial recognition,* dan *voice recognition*. Microsoft Kinect memiliki performansi yang baik, dimensi yang kecil, *open source libraries* yang mampu membuat Microsoft Kinect digunakan pada beberapa *platform OS* (Windows, Linux, dan Mac), serta memiliki harga yang cukup murah (Indrajit, 2012).



Gambar 2.3 Komponen-komponen dari Sensor Microsoft Kinect (Microsoft, 2013)

Sensor Microsoft Kinect mampu mengeluarkan keluaran video dengan *framerate* 60 Hz sebagai kombinasi dari dua sensor yang berbeda, pertama dihasilkan oleh kamera RGB dengan resolusi 8 bit VGA (640x480 piksel) dengan *Bayer Color Filter*, dan yang kedua dihasilkan oleh *monochrome depth sensor* dengan resolusi yang sama dan 2048 tingkat sensitivitas (11 bit).



Gambar 2.4 Blok Diagram Internal dari Sensor Microsoft Kinect (PrimeSense, 2011)

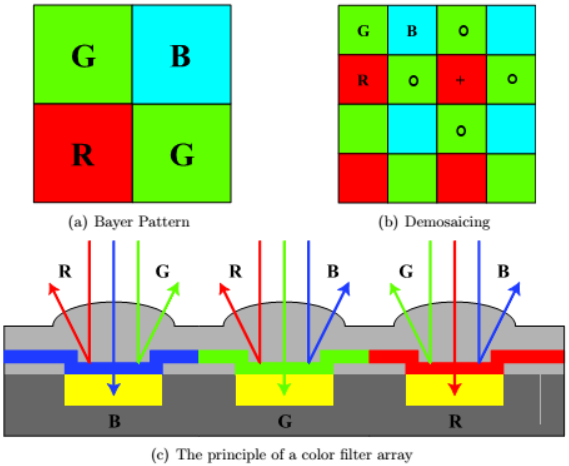
Bagimana sensor-sensor pada Microsoft Kinect terhubung ditunjukkan pada gambar 2.4 Sistem tersebut terdiri dari *depth camera* dan *color camera* yang menggunakan tekologi CMOS serta *microphone* sebagai sensor audio.

2.2.1 Sensor Warna

Semua sensor gambar adalah perangkat *grayscale.* Untuk mendapatkan informasi warna, masing-masing fotodioda dari sensor kamera warna CCD (*charge couple device*) dilapisi dengan filter merah, hijau, atau biru, dalam pola yang berulang. Pola dari filter mengikuti pola Bayer, pola Bayer adalah pola yang sering digunakan untuk akuisisi data digital dari gambar berwarna.

Pola Bayer dari filter warna terlihat seperti gambar 2.5a, dan terus berulang di semua bagian gambar. Pola ini adalah susunan yang umum digunakan dari pola Bayer yang ditunjukkan sebagai GBRG. Dalam pola tersebut, setengah dari jumlah total piksel berwarna hijau (G), sedangkan seperempat dari jumlah total adalah berwarna merah (R) dan biru (B). Karena setiap piksel telah dibuat sensitif hanya untuk satu warna (satu pita spektral),

sensitivitas sensor gambar warna lebih rendah dari sensor monokrom (Puiu, dkk., 2012).

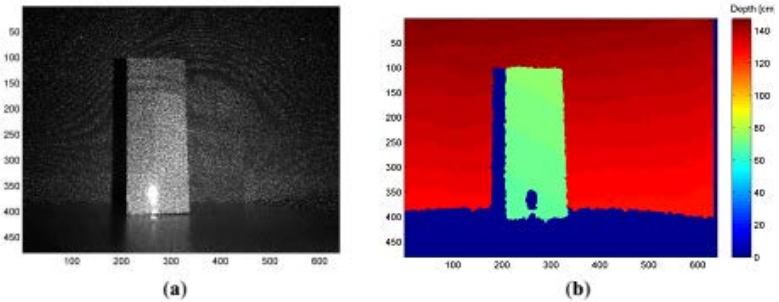


Gambar 2.5 Prinsip Kerja Sensor Warna (Puiu, dkk., 2012)

Untuk mengkonversi gambar dari format Bayer ke format RGB per piksel, dua nilai warna yang hilang dalam setiap piksel perlu diinterpolasi. Nilai warna hijau yang jatuh pada piksel merah atau biru, dihitung dengan nilai rata-rata dari nilai piksel hijau tetangganya dalam pola silang sehingga didapat nilai interpolasi untuk piksel hijau tersebut, begitu pula dengan warna merah (R) dan biru (B).

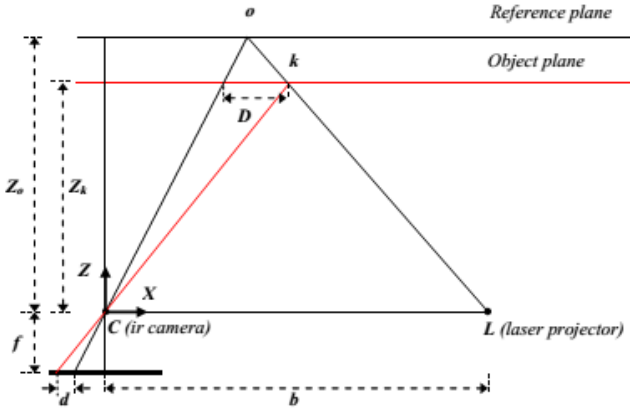
2.2.3 Depth Sensor

Pada Microsoft Kinect terdapat sebuah pemancar inframerah dan sebuah kamera inframerah sebagai sensor kedalaman. Pengukuran kedalaman dapat digambarkan sebagai proses triangulasi. Sumber cahaya inframerah memancarkan sinar tunggal yang dibagi menjadi beberapa sinar oleh kisi difraksi untuk menciptakan pola bintik-bintik konstan yang diproyeksikan ke semua penjuru ruang. Pola ini ditangkap oleh kamera inframerah dan berkorelasi terhadap pola referensi. Pola referensi diperoleh dengan menangkap bidang pada jarak yang dikenal dari sensor, dan disimpan dalam memori sensor.



Gambar 2.6 Hasil Pembacaan Data Kedalamaan Gambar oleh Sensor Microsoft Kinect

Ketika bintik-bintik inframerah diproyeksikan pada sebuah objek yang jaraknya dari sensor lebih kecil atau lebih besar dari bidang referensi, posisi bintik-bintik pada citra inframerah akan digeser ke arah garis dasar antara *infrared emitter* dan pusat perspektif kamera inframerah. Pergeseran ini diukur untuk semua bintik-bintik dengan cara korelasi gambar sederhana, yang menghasilkan disparitas gambar. Untuk setiap piksel, maka jarak ke sensor dapat diambil dari disparitas yang sesuai (Maisto, dkk., 2013).



Gambar 2.7 Skema Pembacaan Data Kedalaman Gambar oleh Sensor Microsoft Kinect (Maisto, dkk., 2013)

Gambar 2.7 menggambarkan hubungan antara jarak dari sebuah objek di titik k ke sensor relatif terhadap bidang referensi dan disparitas terukur d. Untuk menghasilkan koordinat 3D dari titik-titik objek, kita memperhitungkan kedalaman sistem koordinat dengan posisi awalnya, di pusat perspektif kamera inframerah. Sumbu Z ortogonal terhadap bidang gambar ke arah objek, sumbu X tegak lurus terhadap sumbu Z ke arah garis dasar b antara pusat kamera inframerah dan *infrared emitter*, dan sumbu Y ortogonal ke X.

Asumsikan bahwa sebuah objek pada bidang referensi pada jarak Z_o ke sensor, dan titik-titik pada objek ditangkap di bidang gambar kamera inframerah. Jika objek digeser mendekati atau menjauhi sensor, lokasi titik-titik pada bidang gambar akan dipindahkan ke arah X. Hal ini diukur dalam ruang gambar sebagai disparitas d sesuai dengan titik k dalam ruang objek. Dari kesamaan segitiga yang kita miliki maka: $\frac{D}{d} = \frac{Z_0 - Z_k}{Z_0}$ dan $\frac{d}{f} = \frac{D}{Z_k}$ dimana Z_k menunjukkan jarak (kedalaman) dari titik k dalam ruang obyek, b adalah panjang dasar, f adalah panjang fokus kamera inframerah, D adalah perpindahan dari titik k dalam ruang obyek, dan d adalah perbedaan yang diamati pada ruang gambar. Lalu subtitusikan D pada 2 persamaan di atas, maka akan didapat :

$$Z_k = \frac{Z_o}{1 + \frac{Z_0}{fb} d} \tag{2.1}$$

Persamaan diatas merupakan model matematika dasar untuk mencari kedalaman dari perbedaan yang diamati asalkan parameter konstan Z_o, f, dan b dapat ditentukan dengan kalibrasi. Koordinat objek planimetris dari setiap titik kemudian dapat dihitung dengan persamaan berikut :

$$X_k = -\frac{Z_k}{f}(x_k - x_0 + \delta x) \tag{2.2}$$

$$Y_k = -\frac{Z_k}{f}(y_k - y_0 + \delta y) \tag{2.3}$$

dimana x_k dan y_k adalah koordinat gambar titik, x_o dan y_0 adalah koordinat titik awal, dan δ_x dan δ_y adalah koreksi distorsi lensa (Abramov, dkk., 2010).

2.2.4 Kinect SDK

Kinect SDK (*Software Development Kit)* adalah seperangkat *library* yang memungkinkan kita untuk memprogram aplikasi pada berbagai *platform* pengembangan Microsoft menggunakan sensor Microsoft Kinect sebagai masukan. Dengan itu, kita dapat memprogram aplikasi WPF, aplikasi *WinForms*, aplikasi XNA dan aplikasi yang sejenis. Kinect SDK dibuat oleh Microsoft untuk memudahkan para pengembang mengembangkan perangkat Microsoft Kinect untuk berbagai keperluan. Kinect SDK diperlukan untuk dapat membaca data-data mentah dari perangkat Kinect, seperti data kedalaman, data RGB, dan data *skeleton*.

2.3 Candescent NUI

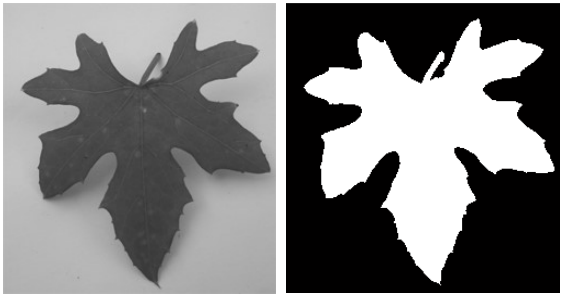
Candescent NUI (CCT NUI) adalah sebuah *library* yang digunakan untuk menghitung informasi tangan secara 3D dari data mentah kedalaman dan RGB yang diperoleh dari Microsoft Kinect SDK. Karena deteksi tangan membutuhkan resolusi yang tinggi maka *library* ini hanya dapat digunakan terbatas pada resolusi 640x480.

Keuntungan utama dari CCT NUI daripada library lain untuk *hand tracking* pada Microsoft Kinect adalah bahwa CCT NUI melakukan komputasi secara *realtime*, sehingga pengguna tidak harus menahan tangan dan jari-jarinya dalam posisi tertentu sampai terdeteksi. Selain itu, CCT NUI tidak memerlukan perangkat keras tambahan selain Microsoft Kinect itu sendiri seperti *graphic card* CUDA pada NVIDIA. Sedangkan kekurangan dari *library* ini adalah CCT NUI rentan terhadap *noise* dan membutuhkan perhitungan yang sangat banyak sehingga memerlukan RAM dengan kapasitas besar.

2.4 Segmentasi

Segmentasi citra merupakan proses yang ditujukan untuk mendapatkan objek-objek yang terkandung di dalam citra atau membagi citra ke dalam beberapa daerah dengan setiap objek yang memiliki kemiripan atribut. Pada citra yang mengandung hanya satu objek, objek dibedakan dari latarbelakangnya (Kadir

& Susanto, 2013). Contoh ditunjukkan pada Gambar 2.8. Pada citra yang mengandung sejumlah objek, proses untuk memilah semua objek akan menjadi lebih kompleks.



(a) Citra daun (b) Hasil segmentasi dalam bentuk biner

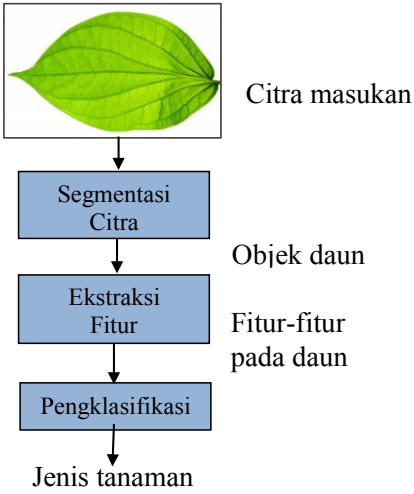
Gambar 2.8 Pemisahan Objek Daun Terhadap Latarbelakang (Kadir & Susanto, 2013)

Gambaran berbagai aplikasi segmentasi serta acuan yang digunakan dapat dilihat pada Tabel 2.1. Secara prinsip, segmentasi dilakukan untuk mendapatkan objek yang diinginkan.

Tabel 2.1 Aplikasi Segmentasi pada Citra

Objek	Citra	Kegunaan Segmentasi	Acuan yang Digunakan
Mobil	Mobil, jalan, dan latarbelakang	Pelacakan mobil	Gerakan dan warna
Struktur permukaan bumi	Foto satelit	Pengklasifikasian area	Tekstur dan warna
Wajah orang	Kerumunan orang	Pengenalan wajah	Warna, bentuk, dan tekstur
Tangan Orang	Tubuh manusia	Pengenalan tangan dan jari	Bentuk, warna, tekstur

Segmentasi juga biasa dilakukan sebagai langkah pertama untuk melakukan proses klasifikasi objek. Setelah segmentasi citra dilaksanakan, fitur yang terdapat pada objek diambil. Sebagai contoh, fitur objek dapat berupa perbandingan lebar dan panjang objek, warna rata-rata objek, atau bahkan tekstur pada objek. Selanjutnya, melalui pengklasifikasi, jenis objek dapat ditentukan.



Gambar 2.9 Segmentasi Sebagai Langkah Pertama Proses Klasifikasi Objek

Berdasarkan teknik yang digunakan, segmentasi dapat dibagi menjadi empat kategori berikut (Rangayyan, 2005) :

a) Teknik peng-ambangan;
b) Metode berbasis batas;
c) Metode berbasis area;
d) Mengkombinasikan kriteria batas dan area.

2.5 Pusat Masa (*Centroid*)

Pusat massa atau *centroid* lazim dilakukan dalam pengolahan citra digital. Pusat massa didefinisikan sebagai lokasi rerata dari intensitas piksel citra yang terdistribusi yang ada pada suatu objek

citra. Pusat massa citra ditemukan dengan menggunakan nilai rerata koordinat setiap piksel yang menyusun objek (Kadir & Susanto, 2013). Pusat massa banyak digunakan untuk memperoleh fitur lebih lanjut. Beberapa contoh dapat dilihat di bawah ini :

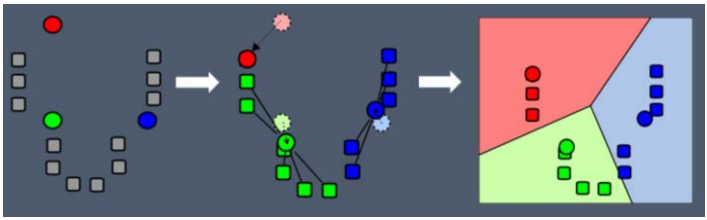
- Pusat massa untuk memperoleh fitur dispersi
- Menghitung jarak terpanjang antara pusat massa dan titik dalam kontur (D_{max}).
- Menghitung jarak terpendek antara pusat massa dan titik dalam kontur (D_{min}).
- Menghitung jarak rata-rata antara pusat massa dan titik dalam kontur (D_{mean}).
- Histogram jarak antara pusat massa dan titik dalam kontur.
- Perbandingan: $\frac{D_{max}}{D_{min}}, \frac{D_{max}}{D_{mean}}, \frac{D_{min}}{D_{mean}}$.

2.6 *K-Means Clustering*

Algoritma *K-means clustering* diterapkan untuk membagi semua piksel ke dalam dua kelompok atau beberapa kelompok yang ditentukan. *K-means clustering* adalah sebuha metode untuk membagi n data ke dalam k kelompok, $C_1, C_2, \dots, C_k$; setiap data dengan rata-rata terdekat, $\mu_1(x, y)$, yang dihitung sebagai rata-rata dari titik-titik di C_i, *K-means clustering* berusaha untuk meminimalkan variasi antar data yang ada di dalam suatu kelompok dan memaksimalkan variasi dengan data yang ada di cluster lainnya (Chang, dkk., 2013).

$$arg \min_{C} \sum_{i=1}^{k} \sum_{p_j(x,y) \in C_i} \left\| p_j(x, y) - \mu_i(x, y) \right\|^2 \quad (2.4)$$

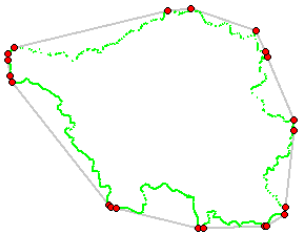
K-means clustering terus dilakukan setiap kali ada perubahan dalam sumber masukan data. Pada awal setiap iterasi, setiap kelompok kosong diinisialisasi dengan titik-titik acak yang terletak dalam rentang nilai ambang tertentu sebagai nilai rata-rata.



Gambar 2.10 Proses Penerapan *K-means Clustering*

2.7 *Convex Hull*

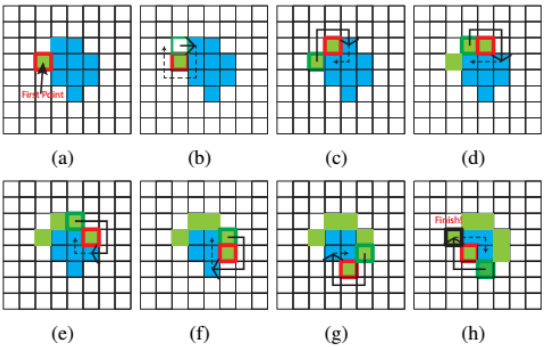
Convex hull didefinisikan sebagai sutau objek dimana garis yang menghubungkan titik-titik di dalam objek tersebut juga berada di dalam objek tersebut. Misal S adalah sekumpulan dari titik-titik $\{p_0, p_1, p_2, \dots, p_n\}$ pada bidang 2D, *convex hull* adalah *convex* poligon terkecil yang berisi semua titik di dalam S. Poligon ini dapat dibayangkan sebagai gelang elastis yang dapat melingkupi tepi objek, hal seperti ini diperlukan untuk keperluan pengenalan objek, dengan menghilangkan tepian objek yang cekung (Graham, 1972).



Gambar 2.11 Contoh C*onvex Hull* (Raheja, dkk., 2011)

2.8 *Contour Tracing*

Countour Tracing merupakan suatu metode yang digunakan untuk mendapatkan tepi objek. Salah satu cara untuk mendapatkan kontur internal suatu objek pada citra yang telah diurutkan menurut letak piksel, yaitu dengan memanfaatkan algoritma pelacakan kontur Moore. Kontur tangan terdeteksi oleh modifikasi algoritma *Moore-Neighbor Tracing*. Setelah pengelompokan, sekelompok titik-titik piksel tangan yang dideteksi lalu disimpan.



Gambar 2.12 Ilustrasi Algoritma *Moore-Neighbor Contour Tracing* (Li, 2012)

Gambar 2.12 menjelaskan bagaimana proses *Moore-Neighbor Contour Tracing* dilakukan. Untuk melacak kontur piksel biru maka dimisalkan piksel hijau adalah piksel kontur yang terdeteksi. Piksel dengan garis merah mewakili piksel kontur saat itu; piksel dengan garis hijau menunjukkan *starting piksel* dari pemeriksaan ketetanggaan, yaitu kontur piksel terakhir yang terdeteksi. garis hitam menunjukkan jalur searah jarum jam dari pemeriksaan ketetanggaan; garis putus-putus menunjukkan alur pemeriksaan yang tidak perlu untuk diselesaikan karena piksel kontur berikutnya telah terdeteksi (Kadir & Susanto, 2013).

2.9 Perhitungan Jarak dan Sudut antara Dua Citra

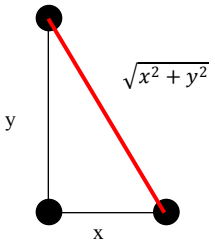
Jarak dan sudut merupakan pendekatan yang umum dipakai pada pengolahan citra digital. Fungsinya adalah untuk menentukan kesamaan atau ketidaksamaan dua vektor fitur. Tingkat kesamaan dinyatakan dengan suatu nilai atau peringkat. Semakin kecil nilai peringkat, semakin dekat kesamaan kedua vektor tersebut.

2.9.1 Jarak *Euclidean*

Jarak *Euclidean* didefinisikan sebagai berikut:

$$j(v_1, v_2) = \sqrt{\sum_{k=1}^{N}(v_1(k) - v_2(k))^2} \qquad (2.5)$$

Dalam hal ini, v_1 dan v_2 adalah dua vektor yang jaraknya akan dihitung dan N menyatakan panjang vektor. Apabila vektor memiliki dua nilai, jarak *Euclidean* dapat digambarkan sebagai sisi miring segitiga (Kadir & Susanto, 2013).



Gambar 2.13 Gambaran Jarak *Euclidean*

Sebagai contoh, terdapat dua vektor seperti berikut :
$v_1 = [\,4, 3, 6]$
$v_2 = [\,2, 3, 7]$
jarak *Euclidean* kedua vektor adalah :
$jarak = \sqrt{(4-2)^2 + (3-3)^2 + (6-7)^2} = \sqrt{5} = 2{,}2361$
Jarak *Euclidean* merupakan jarak yang umum dipakai dalam *image retrieval*.

2.9.2 Jarak *City-Block*

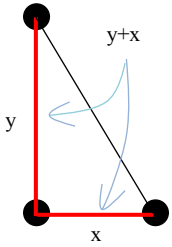
Jarak *City-Block* didefinisikan sebagai berikut:

$$j(v_1, v_2) = \sum_{k=1}^{N} |v_1(k) - v_2(k)| \qquad (2.6)$$

Dalam hal ini, v_1 dan v_2 adalah dua vektor yang jaraknya akan dihitung dan N menyatakan panjang vektor. Apabila vektor memiliki dua nilai, jarak *City-Block* dapat digambarkan sebagai jarak vertikal ditambah jarak horizontal dari vektor pertama ke vektor kedua (Kadir & Susanto, 2013).

Sebagai contoh, terdapat dua vektor seperti berikut :
$v_1 = [\,4, 3, 6]$
$v_2 = [\,2, 3, 7]$

jarak *city-block* kedua vektor tersebut berupa :
$jarak = |4-2| + |3-3| + |6-7| = 3$.



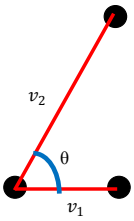
Gambar 2.14 Gambaran Jarak *City-Block*

2.9.3 Sudut Dua Vektor

Sudut dua vektor dapat ditemukan dengan cara berikut :

$$\theta(v_1, v_2) = \tan^{-1}\left[\frac{\sum_{k=1}^{N} v_k(x)}{\sum_{k=1}^{N} v_k(y)}\right] \quad (2.7)$$

Dalam hal ini, v_1 dan v_2 adalah dua vektor yang sudutnya akan dihitung dan N menyatakan panjang vektor. Apabila vektor memiliki dua nilai, sudut anatara dua vektor dapat digambarkan sebagai berikut :



Gambar 2.15 Gambaran Sudut antara 2 Vektor

Sebagai contoh, terdapat dua vektor seperti berikut :
$v_1 = [\,5, 6]$
$v_2 = [\,3, 7]$
sudut antara dua vektor tersebut menjadi :
$Sudut = \tan^{-1}\frac{5+3}{6+7} = 31{,}6\,^{o}$.

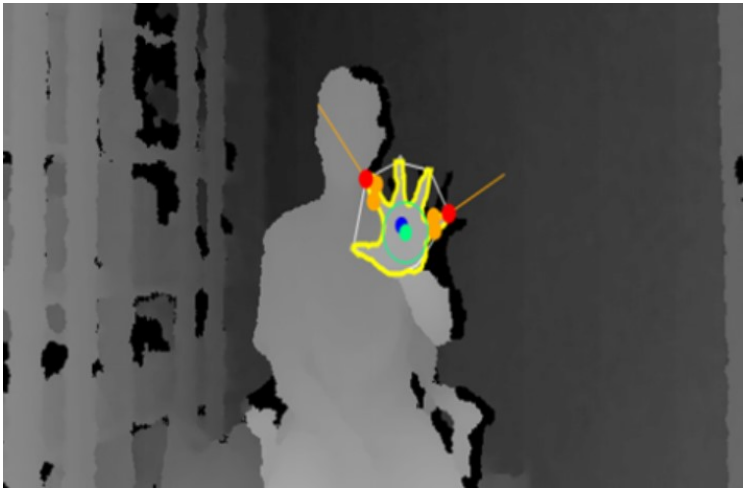
BAB IV
HASIL DAN PEMBAHASAN

Pada bab ini akan dijelaskan mengenai analisa data pada setiap pengujian spesifikasi. Pengujian yang telah dilakukan meliputi :

- Jarak terjauh dimana sistem pengenalan dapat mendeteksi tangan dan mengenali kelima jari.
- Medan Pandang dari Microsoft Kinect (Field Of View)
- Kecepatan komputasi dari sistem pengenalan bahasa isyarat
- Pengaruh perubahan cahaya terhadap penerjemahan bahasa isyarat
- Keakurasian dalam menerjemahkan bahasa isyarat

4.1 Jarak terjauh

Pengujian ini di tujukan untuk menguji spesifikasi sistem pengenalan bahasa isyarat untuk menerjemahkan bahasa isyarat pada jarak terjauh yang masih dapat dideteksi oleh sensor Microsoft Kinect. Dalam pengujian dilakukan 10 kali pengambilan data.



Gambar 4.1 Jarak Terjauh yang Dapat Dideteksi

Gambar 4.1 adalah gambar dari hasil sistem pengenalan bahasa isyarat dapat mendeteksi tangan, namun tidak bisa mendeteksi kelima jari, didapatkan jarak terjauh rata-rata yang diperoleh adalah 1.438 mm seperti yang ditunjukkan oleh tabel 4.1, dimana masih terdapat banyak *noise* yang timbul untuk dapat mendeteksi kelima jari. Dari 10 data yang diambil pendeteksian jari selalu berkedip dan terlihat tidak konstan.

Tabel 4.1 Data Jarak Terjauh yang Dapat Dideteksi

Percobaan ke - i	Jarak Terjauh (mm)
1	1.438
2	1.440
3	1.440
4	1.439
5	1.436
6	1.439
7	1.437
8	1.438
9	1.436
10	1.437
rata-rata	1.438

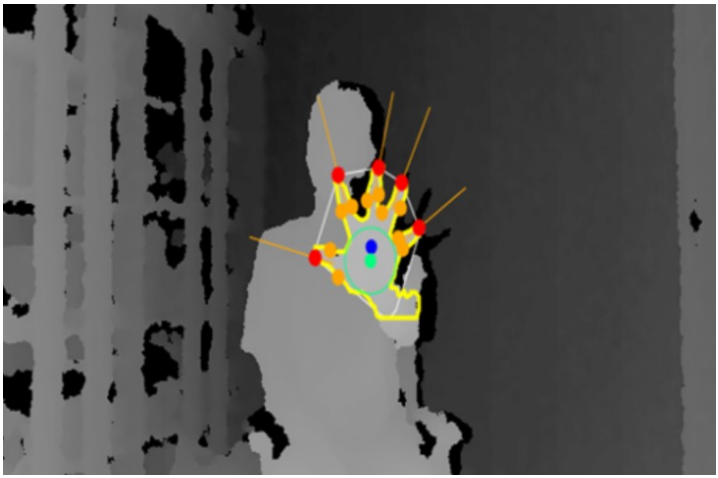
Sedangkan gambar 4.2 adalah gambar ketika sistem pengenalan dapat mendeteksi tangan dan juga mendeteksi kelima jari secara stabil tanpa berkedip atau banyaknya *noise*, didapatkan bahwa jarak terjauh yang bisa dikenali sistem agar dapat mendeteksi tangan dan kelima jari secara utuh dan stabil dengan sedikit *noise* adalah sejauh 1.187,6 mm seperti yang ditunjukkan oleh tabel 4.2.

Dari data jarak terjauh yang diperoleh maka dapat diambil data nilai *threshold* dari jarak kedalaman tangan yang dapat dideteksi, nilai *threshold* ini diterapkan agar hanya data yang stabil dan sedikit noise dari tangan dan kelima jari yang

terdeteksi, sehingga menghindari ketidakpresisian sistem pengenalan bahasa isyarat, nilai *threshold* diterapkan sebesar 1.000-1.200 mm.

Tabel 4.2 Data Jarak Terjauh Sistem Dapat Mengenali 5 Jari

Percobaan ke - i	Jarak Terjauh (mm)
1	1.186
2	1.188
3	1.187
4	1.190
5	1.186
6	1.189
7	1.187
8	1.188
9	1.186
10	1.189
rata-rata	1.187,6

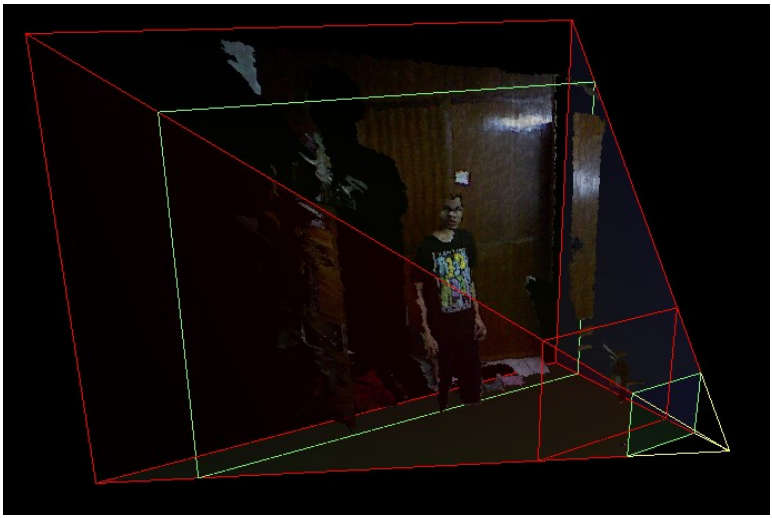


Gambar 4.2 Jarak Terjauh Sistem Dapat Mengenali 5 Jari

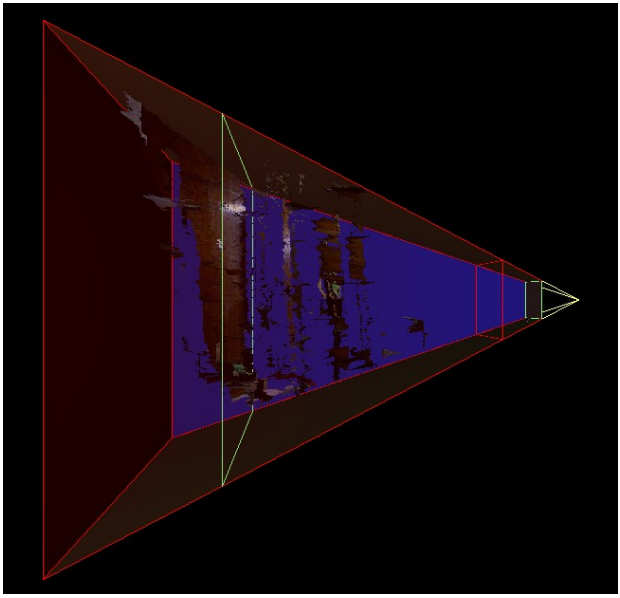
4.2 Medan Pandang (*Field of View*)

Penentuan jarak pandang atau kemampuan *Field Of View* (FOV) kamera menjadi sangat penting karena menentukan tingkat keakurasian dari sistem pengenalan bahasa isyarat dalam menerjemahkan bahasa isyarat. Nilai sudut FOV baik dari pengujian maupun data produk yang diterbitkan microsoft menunjukkan bahwa sudut FOV adalah 58 derajat horizontal dan 45 derajat vertikal.

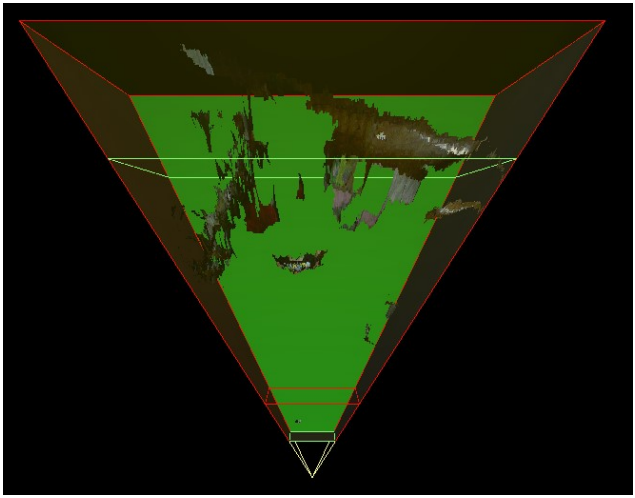
Dari gambar 4.4 , 4.5 , dan 4.6 didapatkan bahwa garis warna merah adalah medan pandang yang dapat dideteksi oleh sensor. Medan pandang berbentuk prisma, sensor memiliki batas untuk dapat mengukur kedalaman yaitu sebesar 3.500 mm sesuai dengan spesifikasi yang didapatkan dimana ditunjukkan pada garis kotak merah yang berada paling belakang, sehingga area yang dapat dikenali oleh sistem pengenalan bahasa isyarat tidak boleh keluar dari area yang dibatasi oleh garis merah, sedangkan garis hijau menunjukkan jangkauan dari sensor Microsoft Kinect dimana sistem dapat mengenali bahasa isyarat dengan optimal.



Gambar 4.3 Gambar Medan Pandang (*Field of View*) Sensor Microsoft Kinect dalam 3 Dimensi



Gambar 4.4 Gambar Medan Pandang (*Field of View*) Sensor Microsoft Kinect Tampak Samping



Gambar 4.5 Gambar Medan Pandang (*Field of View*) Sensor Microsoft Kinect Tampak Atas

4.3 Kecepatan Komputasi

Pengujian kecepatan dilakukan untuk mendapatkan rentang kecepatan sistem dalam melakukan komputasi-komputasi untuk menerjemahkan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia). Percobaan dalam pehitungan waktu komputasi dilakukan dalam menerjemahkan angka 0-9 dalam SIBI.

Tabel 4.3 Tabel Waktu Komputasi

Percobaan ke-i	Waktu Komputasi (ms)
1	3,91
2	3,88
3	3,88
4	3,92
5	3,92
6	3,92
7	3,90
8	3,90
9	3,90
10	3,92
rata-rata	**3,905**

Berdasarkan tabel 4.3 diketahui bahwa waktu komputasi rata-rata yang diperlukan sistem pengenalan bahasa isyarat dengan menggunakan laptop dengan spesifikasi CPU Core i5 2.3 GHz dengan RAM 2 GB DDR3 dan sistem operasi menggunakan Microsoft Windows 8.1 adalah 3,905 ms. Hal ini menunjukkan bahwa kecepatan komputasi yang dilakukan sistem pengenalan bahasa isyarat dilakukan cukup cepat walaupun dengan komputasi dan algoritma yang sangat banyak, sehingga hal ini menunjukkan bahwa sistem pengenalan bahasa isyarat ini cocok untuk diterapkan dalam menerjemahkan bahasa isyarat.

4.4 Pengaruh Pencahayaan

Pengujian ini bertujuan untuk mengetahui rentang tingkat pencahayaan terhadap pembacaan sistem pengenalan bahasa isyarat dalam menerjemahkan bahasa isyarat. Pengukuran dilakukan dengan variasi kuat penerangan 196 lux, 76 lux, 35 lux, 18 lux, 6 lux dan 0 lux (Gelap). Setiap kuat penerangan di ambil 10 kali penerjemahan untuk menerjemahkan angka 0-9 dalam SIBI sehingga total didapat 60 data. Tabel 4.4 berikut menyajikan rata-rata hasil *succes rate* dalam penerjemahan bahasa isyarat.

Tabel 4.4 Pengaruh Kuat Penerangan terhadap *Success Rate*

Kuat Penerangan (Lux)	***Success Rate* (%)**
196	100%
76	100%
35	100%
18	80%
6	80%
0	60%

Dari tabel 4.4 didapatkan bahwa pada semua kondisi cahaya sistem pengenalan bahasa isyarat dapat menerjemahkan bahasa isyarat bahkan saat gelap sekalipun, namun dengan *success rate* yang berbeda-beda. Hal ini disebabkan bahwa dilakukan penggabungan data antara data RGB dan data kedalaman (*depth*) yang didapatkan dari sensor Microsoft Kinect untuk mendapatkan data-data tangan dan jari-jari, walaupun dalam keadaan gelap (0 Lux) sensor masih dapat mengukur data kedalaman pada tangan yang dideteksi yang hanya menggunakan pancaran sinar inframerah (*infrared spekcle*), sehingga kekurangan akan cahaya dapat diatasi dan diminimalisir dan tetap dapat menerjemahkan bahasa isyarat. Didapatkan *succes rate* terendah pada saat gelap yaitu dengan *success rate* sebesar 60% dengan 4 kali kegagalan dalam menerjemahkan bahasa isyarat.

4.5 Keakurasian

Pengujian ini bertujuan untuk menguji keakurasian sistem pengenalan bahasa isyarat dalam menerjemahkan SIBI (Sistem Isyarat Bahasa Indonesia). Bahasa isyarat yang diujikan adalah bahasa isyarat huruf dan angka dalam SIBI yaitu angka 0-9 dan A-Z kecuali huruf J, Z, M, dan N. Kondisi pengujian dilakukan pada kondisi ideal yaitu dengan kondisi cahaya 198 lux dan jarak antara sensor Microsoft Kinect dan tangan sejauh 1.100 mm. Pengujian dilakukan sebanyak 10 kali setiap bahasa isyarat dengan 5 kali menggunakan tangan kiri dan 5 kali dengan tangan kanan.

Tabel 4.5 *Success Rate* dari Sistem Pengenalan Angka dalam SIBI

Bahasa Isyarat	***Succes Rate* (%)**
nol	100%
satu	100%
dua	100%
tiga	100%
empat	100%
lima	100%
enam	100%
tujuh	80%
delapan	80%
sembilan	100%

Dari tabel 4.5 didapatkan bahwa semua bahasa isyarat angka pada SIBI dapat diterjemahkan sepenuhnya, walaupun dengan keakurasian yang berbeda-beda. Dari tabel 4.5 juga menunjukkan bahwa sistem dapat menerjemahkan bahasa isyarat baik dengan tangan kanan maupun tangan kiri. Terlihat bahwa tingkat akurasi yang terendah didapatkan pada saat

menerjemahkan angka “tujuh” dan “delapan” dengan *success rate* 80%.

Tabel 4.6 *Success Rate* dari Sistem Pengenalan Huruf dalam SIBI

Bahasa Isyarat	Succes Rate (%)
A	40%
B	100%
C	80%
D	80%
E	60%
F	100%
G	70%
H	70%
I	100%
K	90%
L	100%
O	30%
P	90%
Q	80%
R	60%
S	30%
T	70%
U	60%
V	100%
W	100%
X	90%
Y	100%

Dari tabel 4.6 juga didapatkan bahwa semua bahasa isyarat huruf pada SIBI dapat diterjemahkan kecuali huruf J, M, N, dan

Z, walaupun dengan keakurasian yang berbeda-beda. Huruf J dan Z tidak dapat diterjemahkan dalam sistem ini, dikarenakan bahasa isyarat J dan Z dalam SIBI diisyaratkan dengan gerakan tangan memmbentuk huruf J dan Z, sehingga metode yang diterapkan dalam sistem ini tidak mampu menerjemahkan bahasa isyarat ini. Sedangkan pada bahasa isyarat huruf M dan N dalam SIBI tidak dapat diterjemahkan karena tidak ada parameter pembeda yang bisa dibedakan dari metode yang diterapkan sehingga kedua bahasa isyarat ini dianggap sama dan tidak bisa diterjemahkan. Dari tabel 4.6 juga menunjukkan bahwa sistem dapat menerjemahkan bahasa isyarat baik dengan tangan kanan maupun tangan kiri. Terlihat bahwa tingkat akurasi yang terendah didapatkan pada saat menerjemahkan huruf "O" dan "S" dengan *success rate* hanya 30%.

Dari semua angka dan huruf yang dapat diterjemahkan oleh sistem pengenalan bahasa isyarat, maka didapat *total success rate* dari sistem pengenalan sebesar 83,75% yang didapatkan dari tabel 4.5 dan 4.6. Sistem pengenalan bahasa isyarat dapat menerjemahkan huruf dan angkan dalam SIBI sebanyak 268 kali dari total pengambila data penerjemahan bahasa isyarat sebanyak 320 kali.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil yang telah didapatkan serta meninjau kembali permasalahan beserta batasan dan tujuan, maka dapat dirumuskan beberapa kesimpulan yang diperoleh dalam pelaksanaan tugas akhir ini, yaitu:

a. Sistem pengenalan bahasa isyarat yang telah dirancang dan dibangun dapat menerjemahkan huruf dan angka dalam SIBI (Sistem Isyarat Bahasa Indonesia) yang telah ditetapkan. Spesifikasi sitem pengenalan bahasa isyarat yang diperoleh yaitu dengan jarak terjauh dimana tangan dapat terdeteksi adalah sejauh 1.438 mm dan jarak terjauh dimana sistem pengenalan dapat mengenali jari-jari adalah sejauh 1.187,6 mm, medan pandang dari sensor Microsoft Kinect sebesar 58 derajat horizontal dan 45 derajat vertikal dan kecepatan komputasi rata-rata adalah 3,905 ms.
b. Sistem pengenalan bahasa isyarat dapat menerjemahkan bahasa isyarat dengan *success rate* terendah dimana sistem pengenalan bahasa isyarat dapat menerjemahkan angka dan huruf dalam SIBI adalah sebesar 30 % pada huruf O dan S dan *total success rate* dari sistem sebesar 83,75%. Sistem juga dapat menerjemahkan bahasa isyarat dalam berbagai kondisi pencahayaan termasuk pada kuat pencahayaan 0 lux (gelap) karena dilengkapi dengan sensor kedalaman dengan *success rate* terendah sebesar 60%.
c. Sistem pengenalan bahasa isyarat yang telah dibangun tidak dapat menerjemahkan isyarat dengan tangan dan jari yang bertumpuk.

5.2 Saran

Dalam penelitian tugas akhir ini, terdapat beberapa hal yang perlu diperbaiki baik dari tinjauan teoritis maupun aplikatif. Oleh

sebab itu, validasi dan penelitian lebih lanjut sangat diperlukan. Saran yang dapat diberikan penulis terkait dengan pengembangan penelitian ini adalah sebagai berikut:

a. Mengganti sensor Microsoft Kinect dengan versi terbaru, versi terbaru yang telah keluar adalah Microsoft Kinect V2.0 yang memiliki sensor yang lebih baik dan algoritma yang lebih canggih.
b. Ditambahkan metode lain seperti *Skeleton Tracking* untuk menambah keakurasian dalam menerjemahkan bahasa isyarat dan juga tidak hanya dapat menerjemahkan angka dan huruf dalam SIBI yang hanya mencakup gerakan tangan tetapi juga kata-kata yang mencakup gerakan lengan.

DAFTAR PUSTAKA

Abdul Muis, Wisnu Indrajit. 2012. Realistic Human Motion Preservation-Imitation Development on Robot with Kinect. ***DGHE (DIKTI)*** Vol.10 (4): 599-608.

Alexey Abramov, Karl Pauwels, Jeremie Papon, Florentin Worg¨ otter¨, and Babette Dellen. 2010. Depth-supported real-time video segmentation with the Kinect. **Institut de Robotica i Inform` atica Industrial (CSIC-UPC) Journa*l*** I: 457-465.

Departemen Pendidikan Nasional Republik Indonesia. 2009. **Pedoman Pelaksanaan Manajemen Sekolah Khusus Tunarungu/Tunawicara (SLB-B)**, Jakarta : Departemen Pendidikan Nasional.

Departemen Pendidikan Nasional Republik Indonesia. 2001. **Kamus Sistem Isyarat Bahasa Indonesia**, Jakarta : Departemen Pendidikan Nasional.

Gerakan untuk Kesejahteraan Tunarungu Indonesia. 2014. **Lebih Lanjut tentang BISINDO**. September 2014. diakses pada 15 Juni, 2015. http://gerkatin.com/detailberita-140-lebih-lanjut-tentang-bisindo.html.

Graham, R.L. 1972. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. **Information Processing Letters** 1: 132-133.

Guochao Chang,Park Jaewan,Oh Chimin,Chilwoo Lee. 2013. A Decision Tree based Real-time Hand Gesture Recognition Method using Kinect. **Journal of Korea Multimedia Society** Vol.16: 1393-140.

Harsh Vardhan Verma, Eshan Aggarwal, Satish Chandra. 2013. Gesture Recognition Using Kinect for Sign Language Translation. *2013* **IEEE Second International Conference on Image Information Processing**. New York.

Iqbal, Mohammad. 2011. **Pengenalan Bahasa Isyarat Indonesia Berbasis Sensor Flex dan Accelerometer Menggunakan**

Dynamic Time Warping. Surabaya: Institut Teknologi Sepuluh Nopember.

Jagdish L. Raheja, Ankit Chaudhary, Kunal Singal. 2011. Tracking of Fingertips and Centres of Palm using KINECT. 2011 **Third International Conference on Computational Intelligence**. Rajasthan.

Kehl, Roland. 2005. **Markerless Motion Capture of Complex Human Movements from Multiple Views**. Zurich: Swiss Federal Institute Of Technology Zurich.

Lee, Johnny. 2009. **Tracking Your Fingers with the Wiimote.** Juli 2014. diakses pada 13 Juni, 2015. http://johnnylee.net/projects/wii/.

Li, Yi. 2012. Multi-scenario Gesture Recognition Using Kinect. **The 17th International Conference on Computer Games.** Louisville.

Marco Maisto, Massimo Panella. 2013. An Accurate Algorithm for the Identification of Fingertips Using an RGB-D Camera. **IEEE Journal on Emerging and Selected Topics in Circuits and System** Vol. 3 (2).

Nazrul Hamizi Adnan, Khairunizam Wan, Shahriman AB. 2012. The Development of a Low Cost Data Glove by Using Flexible Bend Sensor for Resistive Interfaces. **Symposium on Engineering, Science and Business 2012**. Kuala Lumpur.

Puiu, Poenar Daniel. 2012. **Color Sensor and Their Applications**. Nanyang: Nanyang Technological University.

Rayi Yanu Tara, Paulus Insap Santosa, Teguh Bharata Adji. 2012. Hand Segmentation from Depth Image using Anthropometric Approach in Natural Interface Development. **International Journal of Scientific & Engineering Research III** (5): 1-5.

Rizqa Aftoni, Achmad Rizae and Erwin Susanto. 2013. Proportional Derivative Control Based Robot Arm System Using Microsof Kinect. ***2013 International Conference on Robotics, Biomimetics, Intelligent Computational Systems (ROBIONETICS)***. Yogyakarta.

Simarmata, Sesilia Gloria. 2009. **Studi Perilaku dan Kenyamanan dalam Ruang Anak Berkebutuhan Khusus**. Palembang: USU.

Yoshiaki Nakano, Takeo Tatsumi, Kiyoshi Tajitsu. 2009. **Wiimote Positioning System (WPS) - An epoch-making system of indoor position detection.** Tokyo.

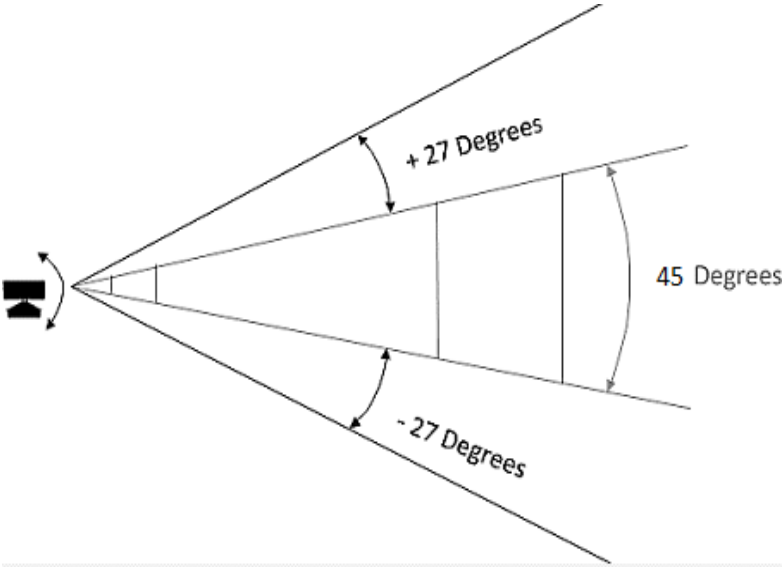
Halaman ini memang dikosongkan

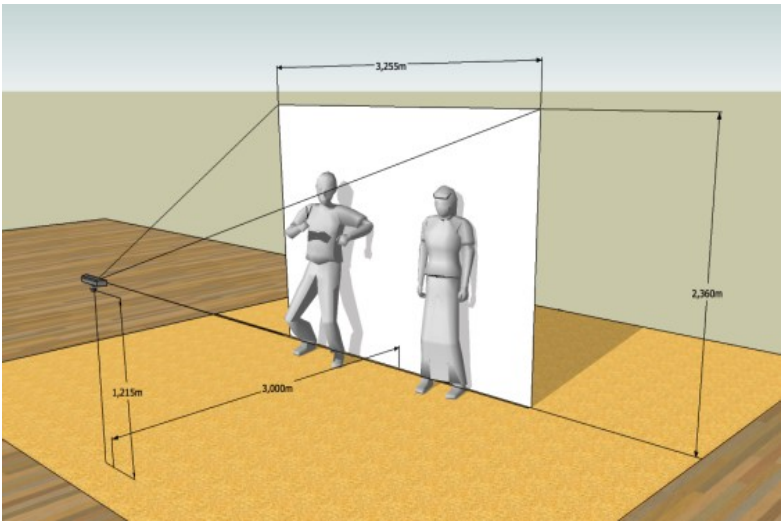
LAMPIRAN A
SPESIFIKASI MICROSOFT KINECT

A.1 Spesifikasi Microsoft Kinect

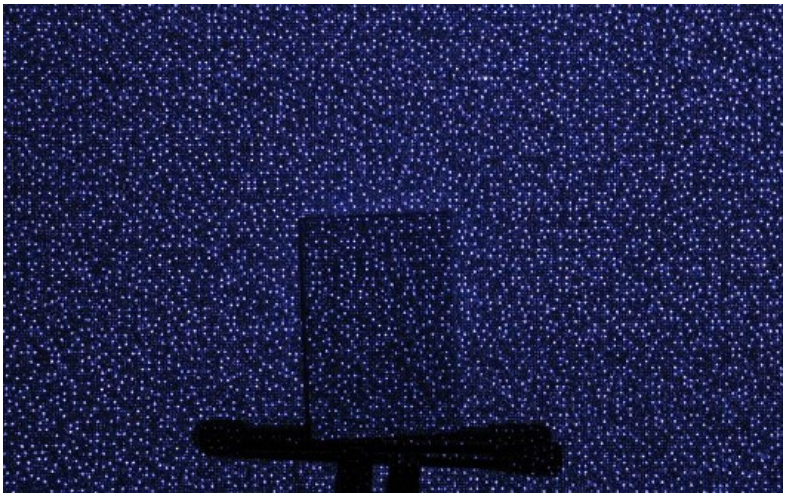
Field of View	*58^{o} H, 45^{o} V*
Distance of Use	*08m - 3.5m*
Sensor	*RGB & Depth*
Image Size	*VGA (640x480) 60 fps*
Dimension	*18 x 3.5 x 5 cm*
Audio Capture	*4-mic array (48KHz)*
Tilt Motor	*Vertical*
Laser Speckle	*830 nm laser diode (60 mW)*
Vertical Tilt Range	*± 27^{o}*
Audio Format	*16 kHz, 24-bit mono code modulation (PCM)*
Audio Input Characteristic	*4 microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression*
Accelerometer Characteristic	*A 2G/4G/8G accelerometer configured for the 2G range, with a 1^{o} accuracy upper limit*

A.2 Medan Pandang (*Field of View*)

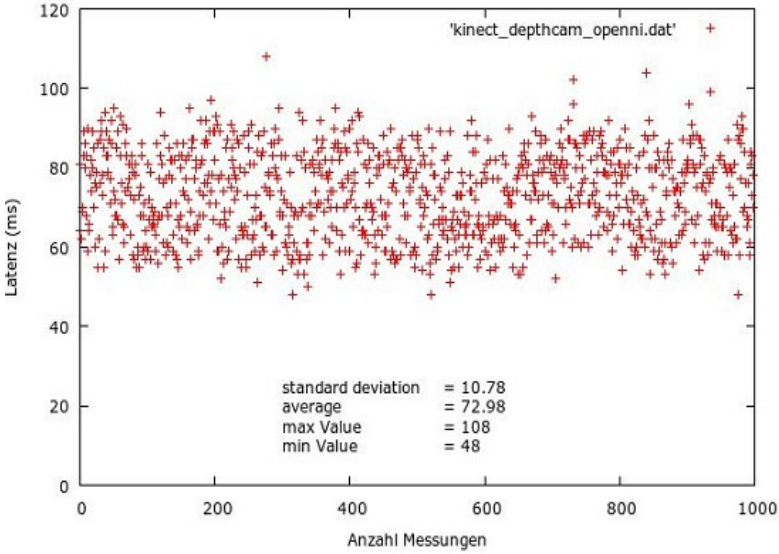




A.3 *Speckle* yang dihasilkan Kinect



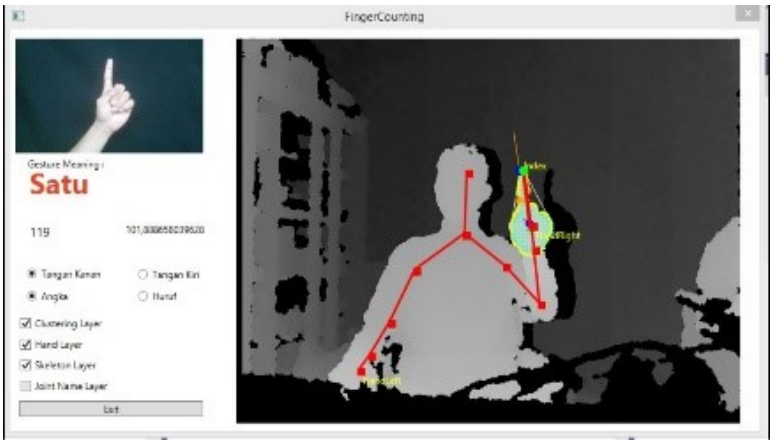
A.4 Latensi dari sensor kedalaman Microsoft Kinect



LAMPIRAN B
HASIL PENGENALAN HURUF DAN ANGKA DALAM SIBI

B.1 Hasil Pengenalan Angka dalam SIBI

Gambar B.1.1 Hasil Pengenalan Angka "Nol"



Gambar B.1.2 Hasil Pengenalan Angka "Satu"

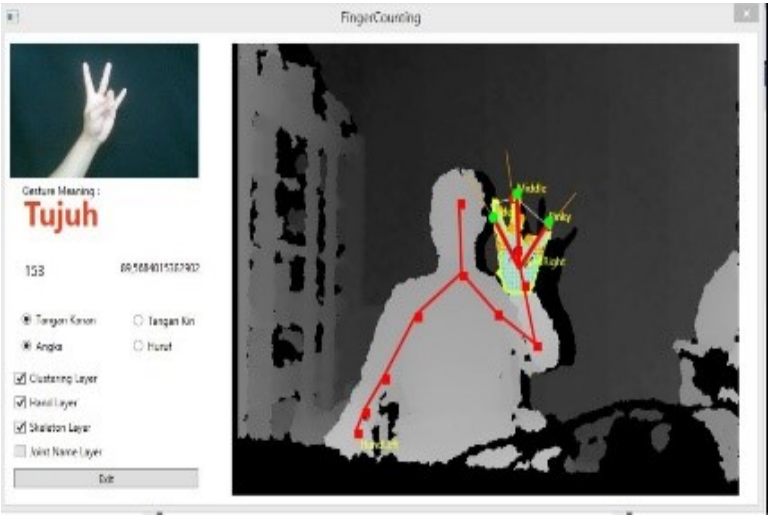
Gambar B.1.3 Hasil Pengenalan Angka “Dua”

Gambar B.1.4 Hasil Pengenalan Angka “Tiga”

Gambar B.1.5 Hasil Pengenalan Angka “Empat”

Gambar B.1.6 Hasil Pengenalan Angka “Lima”

Gambar B.1.7 Hasil Pengenalan Angka “Enam”



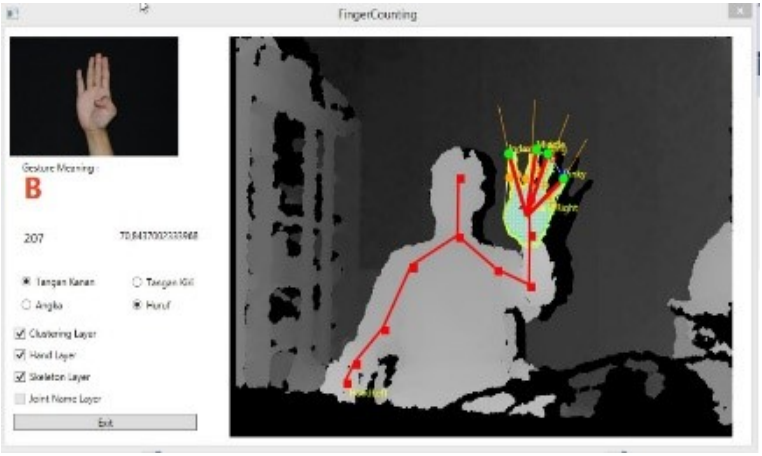
Gambar B.1.8 Hasil Pengenalan Angka “Tujuh”

Gambar B.1.9 Hasil Pengenalan Angka “Delapan”

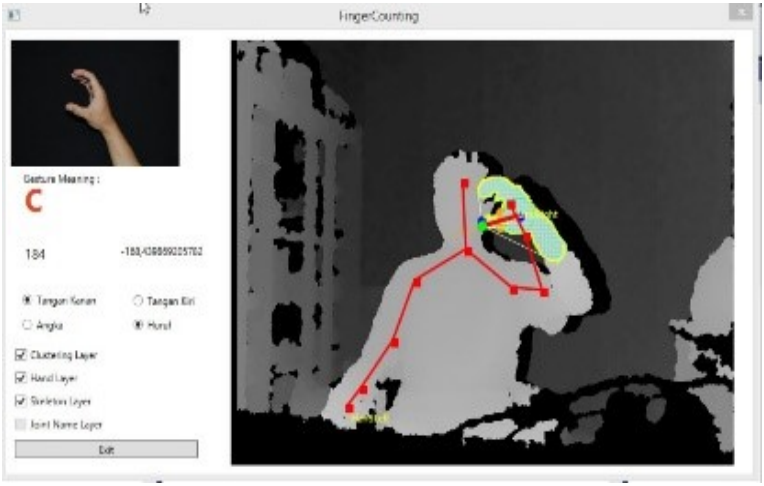
Gambar B.1.10 Hasil Pengenalan Angka “Sembilan”

B.2 Hasil Pengenalan Huruf dalam SIBI

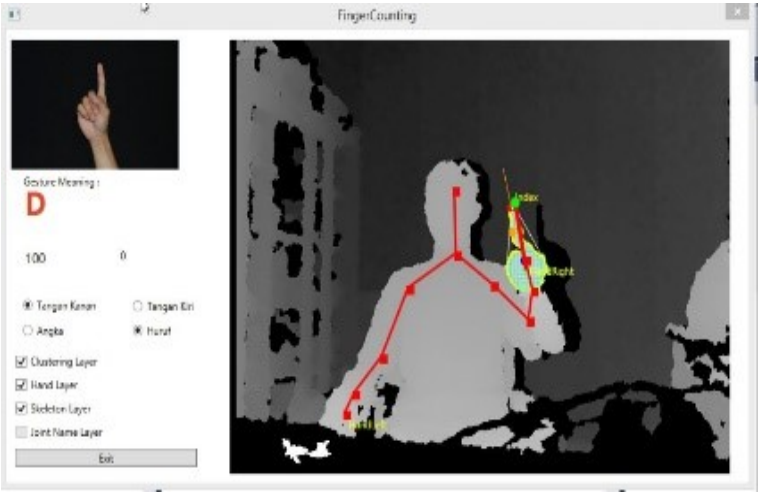
Gambar B.2.1 Hasil Pengenalan Huruf “A”



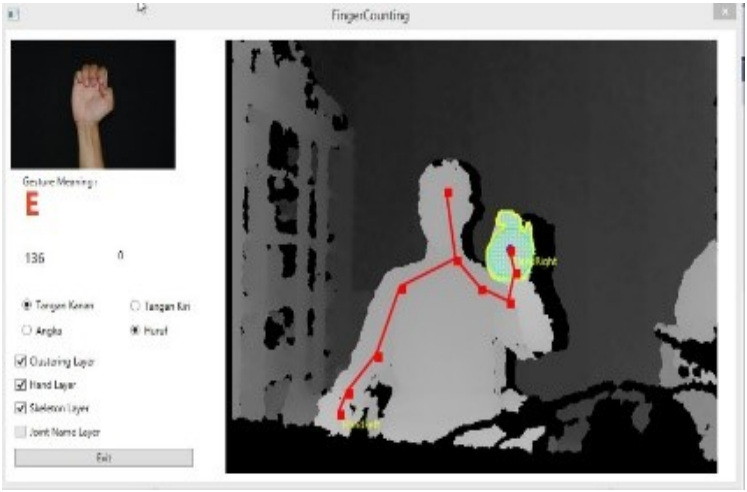
Gambar B.2.2 Hasil Pengenalan Huruf “B”



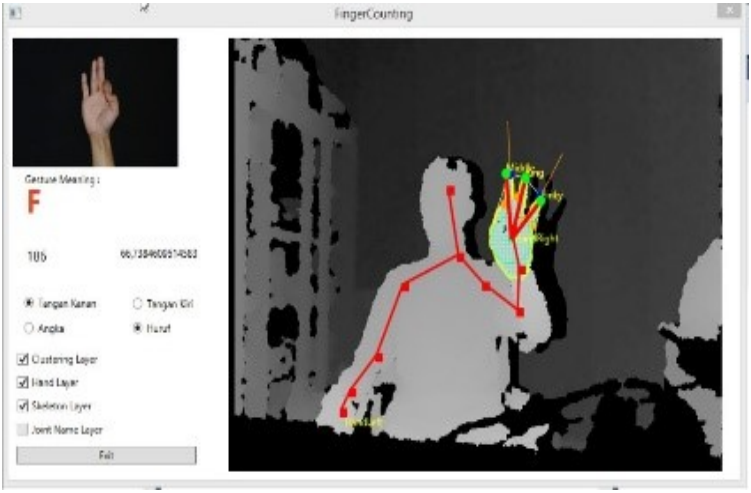
Gambar B.2.3 Hasil Pengenalan Huruf “C”



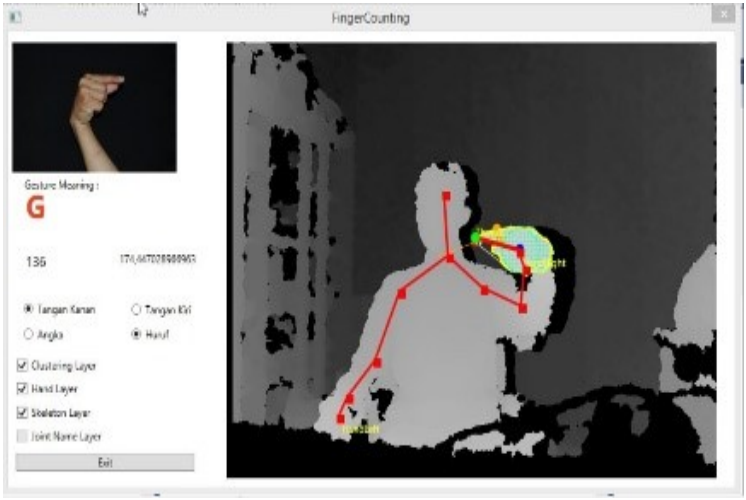
Gambar B.2.4 Hasil Pengenalan Huruf “D”



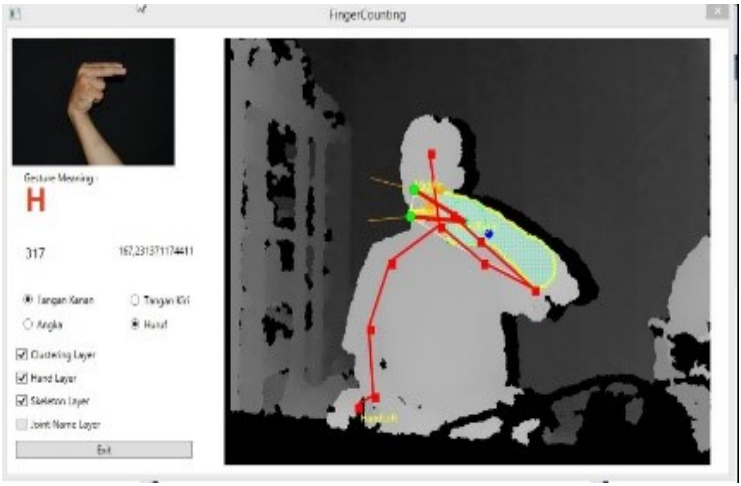
Gambar B.2.5 Hasil Pengenalan Huruf “E”



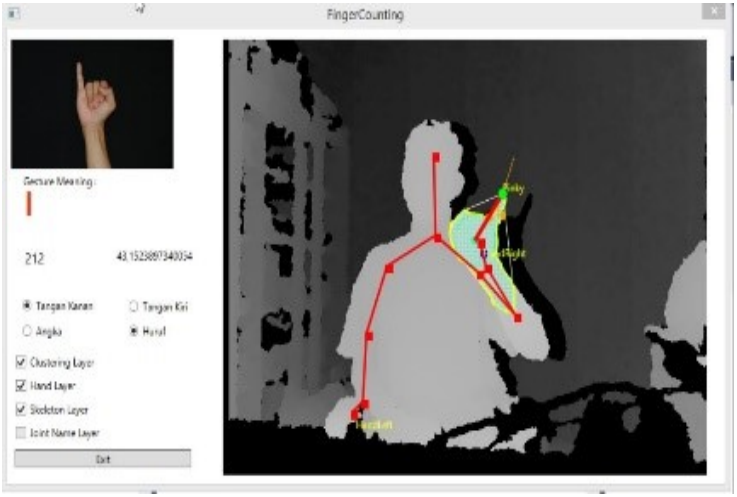
Gambar B.2.6 Hasil Pengenalan Huruf “F”



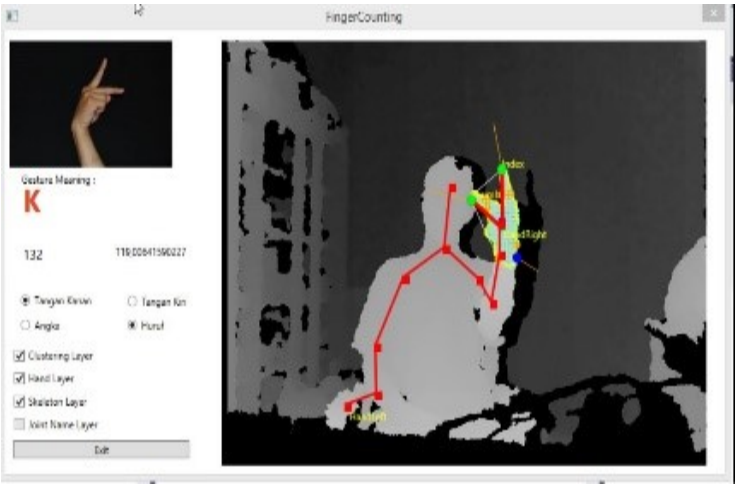
Gambar B.2.7 Hasil Pengenalan Huruf “G”



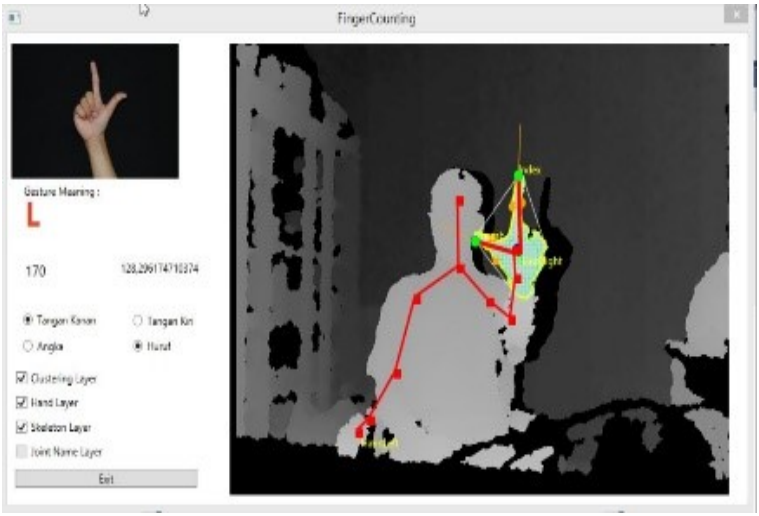
Gambar B.2.8 Hasil Pengenalan Huruf “H”



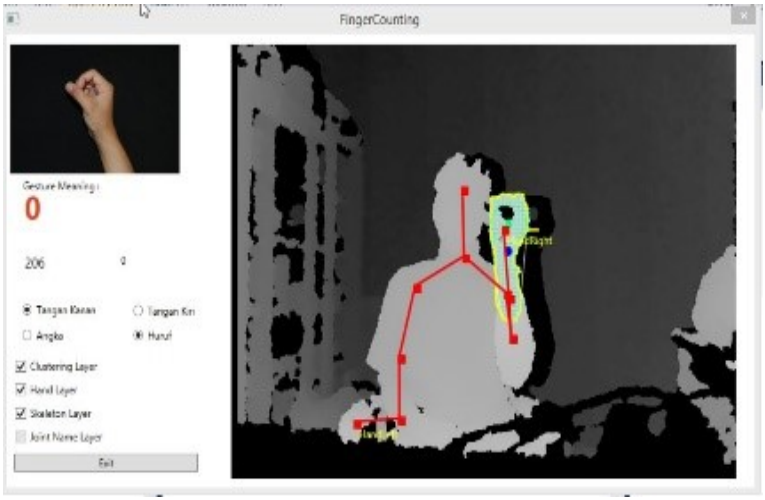
Gambar B.2.9 Hasil Pengenalan Huruf “I”



Gambar B.2.10 Hasil Pengenalan Huruf “K”



Gambar B.2.11 Hasil Pengenalan Huruf “L”



Gambar B.2.12 Hasil Pengenalan Huruf “O”

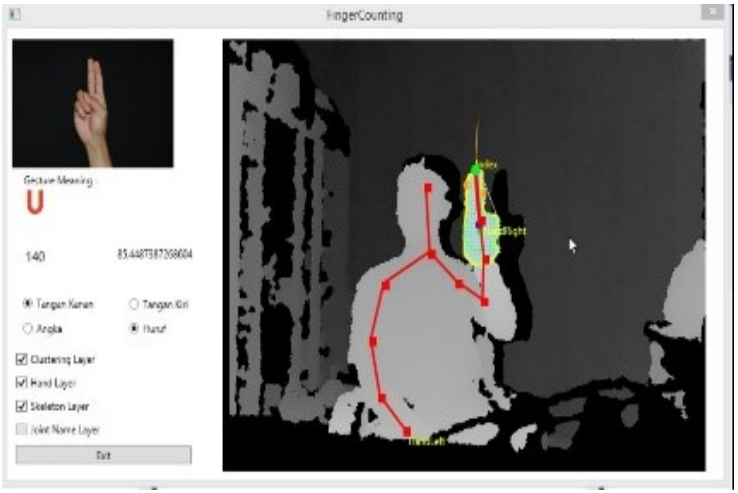
Gambar B.2.13 Hasil Pengenalan Huruf “P”

Gambar B.2.14 Hasil Pengenalan Huruf “Q”

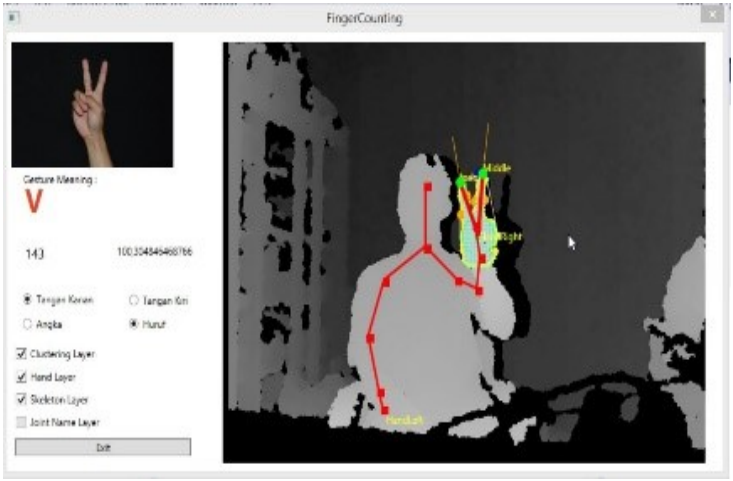
Gambar B.2.15 Hasil Pengenalan Huruf “R”

Gambar B.2.16 Hasil Pengenalan Huruf “S”

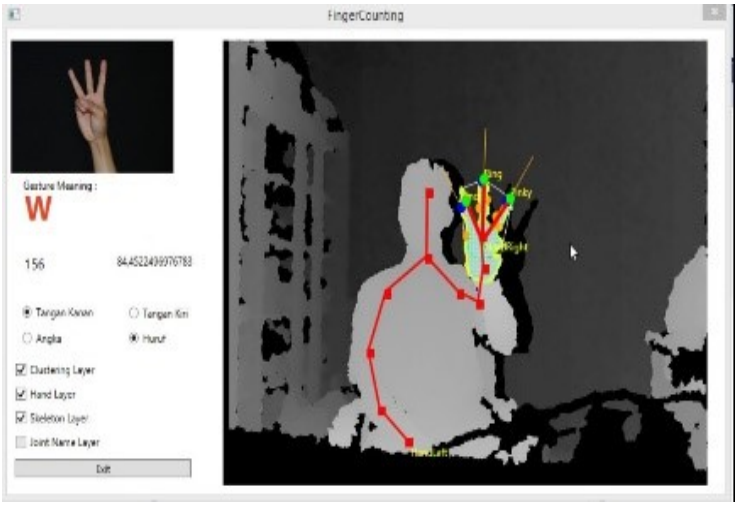
Gambar B.2.17 Hasil Pengenalan Huruf “T”



Gambar B.2.18 Hasil Pengenalan Huruf “U”

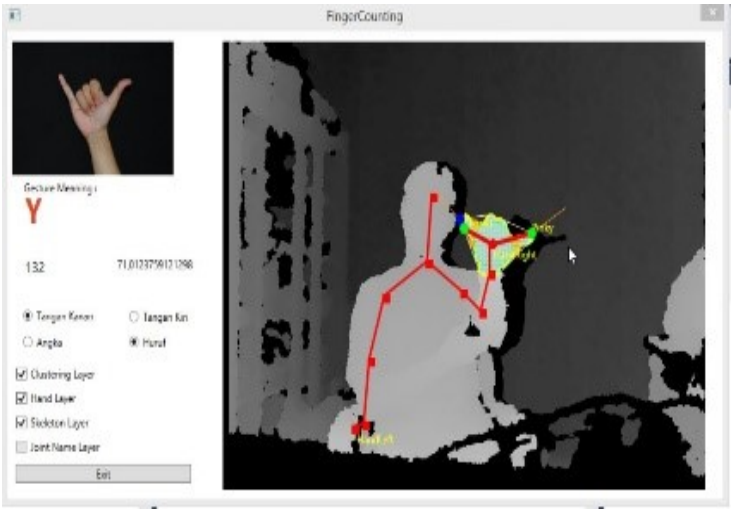


Gambar B.2.19 Hasil Pengenalan Huruf “V”



Gambar B.2.20 Hasil Pengenalan Huruf “W”

Gambar B.2.21 Hasil Pengenalan Huruf “X”



Gambar B.2.22 Hasil Pengenalan Huruf “Y”

LAMPIRAN C
KODE PROGRAM C# PADA SISTEM PENGENALAN BAHASA ISYARAT

Seluruh kode sumber ini dibangun dengan operating system Microsoft Windows 8.1. Kode sumber ini dibangun berdasarkan pustaka Candescent NUI (CCT NUI) yang merupakan pustaka *open source* untuk mendeteksi tangan dan jari-jari, juga digunakan pustaka Kinect SDK untuk mendapatkan data mentah RGB, kedalaman, dan *skeleton* darisensor Microsoft Kinect. Kode sumber yang terlampir disini hanyalah kode sumber yang dibuat dalam sistem pengenalan bahasa isyarat dan bukan termasuk kode bawaan dari pustaka Candescent NUI (CCT NUI) dan Kinect SDK, File-file tersebut adalah:

a. BufferFilter.cs
b. FingerFinder.cs
c. FingerSettings.cs
d. KinectAdapter.cs
e. KinectSettings.cs
f. CountHistory.cs
g. HandInterfaceElement.cs
h. InterfacePainter
i. MainWindow.xaml.cs

Kode-kode tersebut ditulis dalam bahasa C# dan dijalankan pada aplikasi Visual Studio 2013. Sistem pengenalan ini dibuat dalam bentuk format WPF (*Windows Presentation Foundation*) yang hanya bisa dijalankan di sistem operasi Microsoft Windows 8.1.

C.1 Kode File BufferFilter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GRIP
```

```
{
    public class IDStateHolder<T>
    {
        public List<T> NewIDs = new List<T>();
        public List<T> ExistingIDs = new List<T>();
        public List<T> RemovedIDs = new List<T>();
        public List<T> InactiveIDs = new List<T>();
        public void Clear()
        {
            NewIDs = new List<T>();
            ExistingIDs = new List<T>();
            RemovedIDs = new List<T>();
            InactiveIDs = new List<T>();
        }
    }
    public class BufferFilter<T>
    {
        public int EntryBufferTime = 200 * 10000;
        public int ExitBufferTime = 200 * 10000;
        protected Dictionary<T, bool> activeIDs = new
Dictionary<T, bool>();
        protected Dictionary<T, long> startTimes = new
Dictionary<T, long>();
        protected   List<T>   vanishingIDs   =   new
List<T>();
        protected List<T> previousIDs = new List<T>();
        protected  SortedIDHolder<T>  sortedIDHolder  =
new SortedIDHolder<T>();
        protected IDStateHolder<T> idStateHolder = new
IDStateHolder<T>();

        public BufferFilter()
        {
        }

        public  BufferFilter(int  EntryBufferTime,  int
ExitBufferTime)
        {
            this.EntryBufferTime  =  EntryBufferTime  *
10000;
            this.ExitBufferTime   =   ExitBufferTime   *
10000;
        }
```

```
        public IDStateHolder<T> process(IEnumerable<T>
incomingIDs)
        {
            idStateHolder.Clear();

            classifyIDs(incomingIDs);
            processNewIDs(sortedIDHolder.newIDs);

processPendingIDs(sortedIDHolder.pendingIDs);

processRemovedIDs(sortedIDHolder.removedIDs);
            processVanishingIDs();
            previousIDs = new List<T>(incomingIDs);

            return idStateHolder;
        }

        protected  void  processNewIDs(IEnumerable<T>
newIDs)
        {
            foreach (T newID in newIDs)
            {
                if (vanishingIDs.Contains(newID))
                {
                    vanishingIDs.Remove(newID);

idStateHolder.ExistingIDs.Add(newID);
                }
                else
                {
                    startTimes[newID]               =
DateTime.Now.Ticks;
                    activeIDs[newID] = false;

idStateHolder.InactiveIDs.Add(newID);
                }
            }
        }

        protected        void        processPendingIDs
(IEnumerable<T> pendingIDs)
        {
            foreach (T pendingID in pendingIDs)
            {
                if (activeIDs[pendingID])
```

```
idStateHolder.ExistingIDs.Add(pendingID);
                else    if    (DateTime.Now.Ticks    -
startTimes[pendingID] > EntryBufferTime)
                {
                    activeIDs[pendingID] = true;

idStateHolder.NewIDs.Add(pendingID);
                }
                else

idStateHolder.InactiveIDs.Add(pendingID);
            }
        }

        protected                                    void
processRemovedIDs(IEnumerable<T> removedIDs)
        {
            foreach (T removedID in removedIDs)
            {
                if (activeIDs[removedID])
                {
                    vanishingIDs.Add(removedID);
                    startTimes[removedID]            =
DateTime.Now.Ticks;
                }
                else

idStateHolder.InactiveIDs.Add(removedID);
            }
        }

        protected void processVanishingIDs()
        {
            for  (int  i  =  0;  i  <  vanishingIDs.Count;
i++)
            {
                T vanishingID = vanishingIDs[i];
                if        (DateTime.Now.Ticks        -
startTimes[vanishingID] > ExitBufferTime)
                {
                    activeIDs[vanishingID] = false;
                    vanishingIDs.RemoveAt(i);
                    i--;
```

```
idStateHolder.RemovedIDs.Add(vanishingID);
                }
                else

idStateHolder.ExistingIDs.Add(vanishingID);
            }
        }

        protected    void    classifyIDs(IEnumerable<T>
incomingIDs)
        {
            sortedIDHolder.newIDs                    =
incomingIDs.Except(previousIDs);
            sortedIDHolder.pendingIDs                =
incomingIDs.Intersect(previousIDs);
            sortedIDHolder.removedIDs                =
previousIDs.Except(incomingIDs);
        }

        protected class SortedIDHolder<U>
        {
            public IEnumerable<U> newIDs;
            public IEnumerable<U> pendingIDs;
            public IEnumerable<U> removedIDs;
        }
    }
}
```

C.2 Kode File FingerFinder.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using CCT.NUI.Core;
using CCT.NUI.HandTracking;
using CCT.NUI.Core.Clustering;
using CCT.NUI.Core.Shape;
using CCT.NUI.Core.Video;
using CCT.NUI.KinectSDK;
using System.Timers;
using TUIO;
```

```
namespace GRIP
{
    public class FingerFinder
    {
        public int HandBodyDifferenceThreshold;
        public int HandWidth;
        public int HandHeight;
        public int HandLength;
        public int ThumbIndexThreshold;
        public int ThumbMiddleThreshold;
        public int ThumbPinkyThreshold;
        public int IndexPinkyThreshold;
        public int MiddlePinkyThreshold;
        public int RingPinkyThreshold;
        public int MiddleRingThreshold;
        public int IndexRingThreshold;
        public int EntryBufferTime;
        public int ExitBufferTime;
        public int JitterFilterRadius;
        public int SmoothingRadius;
        private double AngleScalingFactor;
        private static float ScaleReciprocalX = 1.0f /
640.0f;
        private static float ScaleReciprocalY = 1.0f /
480.0f;
        private Point wristVector = new Point(0, 0,
0);
        private TuioContact[] contactLookup = new
TuioContact[5];
        private static IComparer<AngularFingerPoint>
fingerComparer = new FingerComparer();
        private Point[] lastKnownFingerPositions = new
Point[5];
        private TuioContact Wrist, Hand, Thumb,
IndexFinger, MiddleFinger, RingFinger, Pinky, Body;
        private KinectAdapter kinect;
        private short[] depthPixelData;
        private bool working = false;
        private BufferFilter<int> filter;
        private AutoResetEvent resetEvent;
        private HandDataSourceSettings handSettings;
        private IHandDataSource handDataSource;
        private HandData handData;
        int firstFingerData;
```

```
        public enum HandType
        {
            Left,
            Right
        }
        public FingerFinder(AutoResetEvent resetEvent,
KinectAdapter      kinect,      HandType      handType,
FingerSettings     settings,     HandDataSourceSettings
handSettingsRoot, IHandDataSource handDataSourceRoot)
        {
            this.kinect = kinect;
            this.handSettings = handSettingsRoot;
            this.handDataSource = handDataSourceRoot;
            if (handType == HandType.Left)
            {
                this.Thumb = kinect.ThumbLeft;
                this.IndexFinger                        =
kinect.IndexFingerLeft;
                this.MiddleFinger                       =
kinect.MiddleFingerLeft;
                this.RingFinger                         =
kinect.RingFingerLeft;
                this.Pinky = kinect.PinkyLeft;
                this.Hand = kinect.HandLeft;
                this.Wrist = kinect.WristLeft;
                AngleScalingFactor = -180.0 / Math.PI;
            }
            else
            {
                this.Thumb = kinect.ThumbRight;
                this.IndexFinger                        =
kinect.IndexFingerRight;
                this.MiddleFinger                       =
kinect.MiddleFingerRight;
                this.RingFinger                         =
kinect.RingFingerRight;
                this.Pinky = kinect.PinkyRight;
                this.Hand = kinect.HandRight;
                this.Wrist = kinect.WristRight;
                AngleScalingFactor = 180.0 / Math.PI;
            }
            this.Body = kinect.Spine;
            this.depthPixelData                         =
kinect.getDepthFrame();
```

```
            this.resetEvent = resetEvent;
            contactLookup   =   new   TuioContact[]   {
this.Thumb,    this.IndexFinger,    this.MiddleFinger,
this.RingFinger, this.Pinky };
            HandBodyDifferenceThreshold              =
settings.HandBodyDifferenceThreshold;
            HandWidth = settings.HandWidth;
            HandHeight = settings.HandHeight;
            HandLength = settings.HandLength;
            ThumbIndexThreshold                      =
settings.ThumbIndexThreshold;
            ThumbMiddleThreshold                     =
settings.ThumbMiddleThreshold;
            ThumbPinkyThreshold                      =
settings.ThumbPinkyThreshold;
            IndexPinkyThreshold                      =
settings.IndexPinkyThreshold;
            MiddlePinkyThreshold                     =
settings.MiddlePinkyThreshold;
            RingPinkyThreshold                       =
settings.RingPinkyThreshold;
            MiddleRingThreshold                      =
settings.MiddleRingThreshold;
            IndexRingThreshold                       =
settings.IndexRingThreshold;
            EntryBufferTime                          =
settings.EntryBufferTime;
            ExitBufferTime = settings.ExitBufferTime;
            JitterFilterRadius                       =
settings.JitterFilterRadius;
            SmoothingRadius                          =
settings.SmoothingRadius;
            filter                =                new
BufferFilter<int>(EntryBufferTime,    ExitBufferTime);
handSettings.MinimumDistanceBetweenFingerPoints      =
settings.MinimumDistanceBetweenFingerPoints;

handSettings.MinimumDistanceIntersectionPoints       =
settings.MinimumDistanceIntersectionPoints;

handSettings.MinimumDistanceFingerPointToIntersectionL
ine                                                  =
settings.MinimumDistanceFingerPointToIntersectionLine;
```

```
handSettings.MaximumDistanceBetweenIntersectionPoints
= settings.MaximumDistanceBetweenIntersectionPoints;
            this.handDataSource.NewDataAvailable    +=
new NewDataHandler<HandCollection>(fingersReady);
        }
        public void processHand(object threadContext)
        {

        }
        private void fingersReady(HandCollection data)
        {
                Dictionary<int,    AngularFingerPoint>
incomingFingerLookupLeft                                =
processIncomingFingers(data);
                IDStateHolder<int>
processedFingerIDsLeft                                  =
filter.process(incomingFingerLookupLeft.Keys);


assignNewFingers(processedFingerIDsLeft.NewIDs,
incomingFingerLookupLeft);

updateExistingFingers(processedFingerIDsLeft.ExistingI
Ds, incomingFingerLookupLeft);

unassignRemovedFingers(processedFingerIDsLeft.RemovedI
Ds);
            resetEvent.Set();
        }
        private   Dictionary<int,   AngularFingerPoint>
processIncomingFingers(HandCollection data)
        {
            Dictionary<int,         AngularFingerPoint>
incomingFingerLookup     =     new     Dictionary<int,
AngularFingerPoint>(5);
            if (data.HandsDetected)
            {
                int handX = (int)(Hand.xpos * 640);
                int handY = (int)(Hand.ypos * 480);
                wristVector.X = (Wrist.xpos * 640) -
(data.Hands[0].PalmX * 640);
                wristVector.Y = (Wrist.ypos * 480) -
(data.Hands[0].PalmY * 480);
```

```
                IList<FingerPoint>    fingerPoints    =
data.Hands[0].FingerPoints;
                List<AngularFingerPoint>
augmentedFingerPoints               =               new
List<AngularFingerPoint>(fingerPoints.Count);
                for    (int    i    =    0;    i    <
fingerPoints.Count; i++)
                {
                    FingerPoint        finger        =
fingerPoints[i];
                    Point location = finger.Location;
                    AngularFingerPoint augmentedFinger
= new AngularFingerPoint(location, finger.Id);
                    augmentedFinger.angle            =
calculateAngle(wristVector,  new  Point((location.X  *
640) - handX, (location.Y * 480) - handY, 0));

augmentedFingerPoints.Add(augmentedFinger);
                }
augmentedFingerPoints.Sort(fingerComparer);

                if (augmentedFingerPoints.Count == 1)
                {
                    if (augmentedFingerPoints[0].angle
<= 130)
                    {
                        augmentedFingerPoints[0].Id  =
0;
                        incomingFingerLookup.Add(0,
augmentedFingerPoints[0]);
                    }
                    else                                if
(augmentedFingerPoints[0].angle      >      130      &&
augmentedFingerPoints[0].angle <= 185)
                    {
                        augmentedFingerPoints[0].Id  =
1;
                        incomingFingerLookup.Add(1,
augmentedFingerPoints[0]);
                    }
                    else                                if
(augmentedFingerPoints[0].angle      >      185      &&
augmentedFingerPoints[0].angle <= 215)
                    {
```

```
                        augmentedFingerPoints[0].Id  =
2;
                        incomingFingerLookup.Add(2,
augmentedFingerPoints[0]);
                    }
                    else                              if
(augmentedFingerPoints[0].angle     >     215     &&
augmentedFingerPoints[0].angle <= 225)
                    {
                        augmentedFingerPoints[0].Id  =
3;
                        incomingFingerLookup.Add(3,
augmentedFingerPoints[0]);
                    }
                    else
                    {
                        augmentedFingerPoints[0].Id  =
4;
                        incomingFingerLookup.Add(4,
augmentedFingerPoints[0]);
                    }
                }
                else  if  (augmentedFingerPoints.Count
== 2)
                {
                    AngularFingerPoint  firstPoint  =
augmentedFingerPoints[0];
                    AngularFingerPoint  secondPoint  =
augmentedFingerPoints[1];

                    double diff = secondPoint.angle -
firstPoint.angle;
                    if   (firstPoint.angle   <=   100)
//thumb
                    {
                        if(diff                       <
ThumbMiddleThreshold) //thumb and index
                        {
                            firstPoint.Id = 0;
                            secondPoint.Id = 1;

incomingFingerLookup.Add(0, firstPoint);

incomingFingerLookup.Add(1, secondPoint);
```

```
                        }
                        else      if      (diff      >
ThumbIndexThreshold  &&  diff  <  ThumbPinkyThreshold)
//thumb and middle
                        {
                            firstPoint.Id = 0;
                            secondPoint.Id = 2;

incomingFingerLookup.Add(0, firstPoint);

incomingFingerLookup.Add(2, secondPoint);

                        }
                        else //thumb and pinky
                        {
                            firstPoint.Id = 0;
                            secondPoint.Id = 4;

incomingFingerLookup.Add(0, firstPoint);

incomingFingerLookup.Add(4, secondPoint);
                        }
                    }
                    else if (firstPoint.angle > 130 &&
firstPoint.angle <= 185) //index
                    {
                        if           (diff           <
IndexPinkyThreshold) //index and middle
                        {
                            firstPoint.Id =1;
                            secondPoint.Id = 2;

incomingFingerLookup.Add(1, firstPoint);

incomingFingerLookup.Add(2, secondPoint);
                        }
                        else //index and pinky
                        {
                            firstPoint.Id = 1;
                            secondPoint.Id = 4;

incomingFingerLookup.Add(1, firstPoint);

incomingFingerLookup.Add(4, secondPoint);
                        }
```

```
                    }
                    else if (firstPoint.angle > 185 &&
firstPoint.angle <= 215)//middle and pinky
                    {
                        firstPoint.Id = 2;
                        secondPoint.Id = 4;
                        incomingFingerLookup.Add(2,
firstPoint);
                        incomingFingerLookup.Add(4,
secondPoint);
                    }
                    else if (firstPoint.angle > 215 &&
firstPoint.angle <= 225) //ring and pinky
                    {
                        firstPoint.Id = 3;
                        secondPoint.Id = 4;
                        incomingFingerLookup.Add(3,
firstPoint);
                        incomingFingerLookup.Add(4,
secondPoint);
                    }
                }
                else  if  (augmentedFingerPoints.Count
== 3)
                {
                    AngularFingerPoint  firstPoint  =
augmentedFingerPoints[0];
                    AngularFingerPoint  secondPoint  =
augmentedFingerPoints[1];
                    AngularFingerPoint  thirdPoint  =
augmentedFingerPoints[2];

                    double diff1 = secondPoint.angle -
firstPoint.angle;
                    double diff2 = thirdPoint.angle -
secondPoint.angle;

                    if   (firstPoint.angle   <=   130)
//thumb
                    {
                        //thumb and index
                        firstPoint.Id = 0;
                        secondPoint.Id = 1;
                        incomingFingerLookup.Add(0,
firstPoint);
```

```
                        incomingFingerLookup.Add(1,
secondPoint);

                        if(diff2                        <
IndexPinkyThreshold) // thum, index, middle
                        {
                            thirdPoint.Id = 2;

incomingFingerLookup.Add(2, thirdPoint);
                        }
                        else // thumb, index, pinky
                        {
                            thirdPoint.Id = 4;

incomingFingerLookup.Add(4, thirdPoint);
                        }
                    }
                    else if (firstPoint.angle > 130 &&
firstPoint.angle <= 185) //index
                    {
                        firstPoint.Id = 1;
                        incomingFingerLookup.Add(1,
firstPoint);
                        if           (diff1           <
IndexRingThreshold) //index and middle
                        {
                            secondPoint.Id = 2;

incomingFingerLookup.Add(2, secondPoint);
                            if  (diff2  >=  50)//index,
middle and pinky
                            {
                                thirdPoint.Id = 4;

incomingFingerLookup.Add(4, thirdPoint);
                            }
                            else  //index,  middle  and
ring
                            {
                                thirdPoint.Id = 3;

incomingFingerLookup.Add(3, thirdPoint);
                            }
                        }
```

```
                        else     if     (diff1     >=
IndexRingThreshold) //index, ring and pinky
                        {
                            secondPoint.Id = 3;
                            thirdPoint.Id = 4;

incomingFingerLookup.Add(3, secondPoint);

incomingFingerLookup.Add(4, thirdPoint);
                        }
                    }
                    else //middle, ring, pinky
                    {
                        firstPoint.Id = 2;
                        secondPoint.Id = 3;
                        thirdPoint.Id = 4;
                        incomingFingerLookup.Add(2,
firstPoint);
                        incomingFingerLookup.Add(3,
secondPoint);
                        incomingFingerLookup.Add(4,
thirdPoint);
                    }
                }
                else  if  (augmentedFingerPoints.Count
== 4)
                {
                    AngularFingerPoint  firstPoint  =
augmentedFingerPoints[0];
                    AngularFingerPoint  secondPoint  =
augmentedFingerPoints[1];
                    AngularFingerPoint  thirdPoint  =
augmentedFingerPoints[2];
                    AngularFingerPoint  fourthPoint  =
augmentedFingerPoints[3];

                    double diff = secondPoint.angle -
firstPoint.angle;

                    if   (firstPoint.angle   <=   130)
//thumb
                    {
                        firstPoint.Id = 0;
                        incomingFingerLookup.Add(0,
firstPoint);
```

```
                        if(diff < ThumbMiddleThreshold)
//thumb, index, middle and pinky
                        {
                            secondPoint.Id = 1;
                            thirdPoint.Id = 2;
                            fourthPoint.Id = 4;
                            incomingFingerLookup.Add(1,
secondPoint);
                            incomingFingerLookup.Add(2,
thirdPoint);
                            incomingFingerLookup.Add(4,
fourthPoint);
                        }
                        else //thumb, middle, ring and
pinky
                        {
                            secondPoint.Id = 2;
                            thirdPoint.Id = 3;
                            fourthPoint.Id = 4;
                            incomingFingerLookup.Add(2,
secondPoint);
                            incomingFingerLookup.Add(3,
thirdPoint);
                            incomingFingerLookup.Add(4,
fourthPoint);
                        }
                    }
                    else if (firstPoint.angle > 130 &&
firstPoint.angle <= 185) //index, middle, ring and
pinky
                    {
                        firstPoint.Id = 1;
                        secondPoint.Id = 2;
                        thirdPoint.Id = 3;
                        fourthPoint.Id = 4;
                        incomingFingerLookup.Add(1,
firstPoint);
                        incomingFingerLookup.Add(2,
secondPoint);
                        incomingFingerLookup.Add(3,
thirdPoint);
                        incomingFingerLookup.Add(4,
fourthPoint);
                    }
```

```
                }
                else if (augmentedFingerPoints.Count
== 5)
                {
                    AngularFingerPoint firstPoint =
augmentedFingerPoints[0];
                    AngularFingerPoint secondPoint =
augmentedFingerPoints[1];
                    AngularFingerPoint thirdPoint =
augmentedFingerPoints[2];
                    AngularFingerPoint fourthPoint =
augmentedFingerPoints[3];
                    AngularFingerPoint fiftPoint =
augmentedFingerPoints[4];

                    firstPoint.Id = 0;
                    secondPoint.Id = 1;
                    thirdPoint.Id = 2;
                    fourthPoint.Id = 3;
                    fiftPoint.Id = 4;
                    incomingFingerLookup.Add(0,
firstPoint);
                    incomingFingerLookup.Add(1,
secondPoint);
                    incomingFingerLookup.Add(2,
thirdPoint);
                    incomingFingerLookup.Add(3,
fourthPoint);
                    incomingFingerLookup.Add(4,
fiftPoint);
                }
            }
                return incomingFingerLookup;
        }

        private    void    assignNewFingers(List<int>
newFingerPointIDs, Dictionary<int, AngularFingerPoint>
newFingerPoints)
        {
            foreach (int Id in newFingerPointIDs)
            {

updateFingerLocation(newFingerPoints[Id]);

kinect.runContactAddedListeners(contactLookup[Id]);
```

```
            }
        }
        private void updateExistingFingers(List<int>
updatedFingerPointIDs,                  Dictionary<int,
AngularFingerPoint> updatedFingerPoints)
        {
            foreach (int Id in updatedFingerPointIDs)
            {
                AngularFingerPoint finger;
                if
(updatedFingerPoints.TryGetValue(Id, out finger))
                {
                    Point        oldLocation        =
lastKnownFingerPositions[Id];
                    float   deltaX   =   finger.X   -
oldLocation.X;
                    float   deltaY   =   finger.Y   -
oldLocation.Y;
                    float   deltaZ   =   finger.Z   -
oldLocation.Z;
                    float            dist           =
(float)Math.Sqrt(deltaX * deltaX + deltaY * deltaY +
deltaZ * deltaZ);
                    // This move has to be big enough
to not be considered noise
                    if (dist > JitterFilterRadius)
                    {
                        // If this move is bigger than
the smoothing radius, trim it down
                        if (dist > SmoothingRadius)
                        {
                            float   scalingFactor   =
SmoothingRadius / dist;
                            float newX = oldLocation.X
+ deltaX * scalingFactor;
                            float newY = oldLocation.Y
+ deltaY * scalingFactor;
                            float newZ = oldLocation.Z
+ deltaZ * scalingFactor;
                            updateFingerLocation(Id,
newX, newY, newZ);
                        }
                        else
```

```
updateFingerLocation(finger);
kinect.runContactChangedListeners(contactLookup[Id]);
                    }
                }
            }
        }
        private void unassignRemovedFingers(List<int>
removedFingerPointIDs)
        {
            foreach (int Id in removedFingerPointIDs)
            {
                contactLookup[Id].active = false;

kinect.runContactRemovedListeners(contactLookup[Id]);
            }
        }
        private                                    void
updateFingerLocation(AngularFingerPoint finger)
        {
            int Id = finger.Id;
            TuioContact contact = contactLookup[Id];
            contact.active = true;
            contact.xpos       =       finger.X       *
ScaleReciprocalX;
            contact.ypos       =       finger.Y       *
ScaleReciprocalY;
            contact.zpos = finger.Z;
            lastKnownFingerPositions[Id]             =
finger.Location;
        }
        private   void   updateFingerLocation(int   Id,
float newX, float newY, float newZ)
        {
            TuioContact contact = contactLookup[Id];
            contact.active = true;
            contact.xpos = newX * ScaleReciprocalX;
            contact.ypos = newY * ScaleReciprocalY;
            contact.zpos = newZ;
            lastKnownFingerPositions[Id]     =     new
Point(newX, newY, newZ);
        }
        public bool IsRunning
        {
            get
```

```
            {
                return true;
            }
        }
        private     double     calculateDistance(Point
finger1Vector, Point finger2Vector)
        {
            double              distance             =
CCT.NUI.Core.Point.Distance2D(finger1Vector,finger2Vec
tor);
            return distance;
        }
        private      double      calculateAngle(Point
finger1Vector, Point finger2Vector)
        {
            double   angle   =   AngleScalingFactor   *
Math.Atan2(
                finger1Vector.X  *  finger2Vector.Y  -
finger1Vector.Y * finger2Vector.X,
                finger1Vector.X  *  finger2Vector.X  +
finger1Vector.Y * finger2Vector.Y);
            return angle < 0 ? angle + 360 : angle;
        }
        private class AngularFingerPoint : FingerPoint
        {
            public double angle;

            public AngularFingerPoint(Point point, int
Id)
                : base(point)
            {
                this.Id = Id;
            }
        }
        private      class      FingerComparer      :
IComparer<AngularFingerPoint>
        {
            public  int  Compare(AngularFingerPoint  a,
AngularFingerPoint b)
            {
                return a.angle.CompareTo(b.angle);
            }
        }
    }
}
```

C.3 Kode File FingerSettings.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GRIP
{
    public class FingerSettings : ISettingManager
    {
        public int HandBodyDifferenceThreshold = 200;
        public int HandWidth = 70;
        public int HandHeight = 70;
        public int HandLength = 60;
        public int ThumbIndexThreshold = 60;
        public int ThumbMiddleThreshold = 90;
        public int ThumbPinkyThreshold = 120;
        public int IndexPinkyThreshold = 50;
        public int MiddlePinkyThreshold = 50;
        public int RingPinkyThreshold = 30;
        public int MiddleRingThreshold = 30;
        public int IndexRingThreshold = 50;
        public int EntryBufferTime = 200;
        public int ExitBufferTime = 200;
        public int JitterFilterRadius = 10;
        public int SmoothingRadius = 30;
        public                                    float
MinimumDistanceBetweenFingerPoints = 22;
        public float MinimumDistanceIntersectionPoints
= 18;
        public                                    float
MinimumDistanceFingerPointToIntersectionLine = 16;
        public                                    float
MaximumDistanceBetweenIntersectionPoints = 23;
    }
}
```

C.4 Kode File KinectAdapter.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Windows;
```

```
using System.Threading;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Shapes;
using System.Windows.Threading;
using Microsoft.Kinect;
using TUIO;
using CCT.NUI.HandTracking;

namespace GRIP
{
    public class KinectAdapter : IAdapter
    {
        readonly KinectSensor sensor;
        readonly Canvas canvas;
        Skeleton skeleton;
        private short[] depthPixelData;
        private int depthFrameWidth;
        private int realDepth;
        private DepthImageFrame DepthFrameData;
        public bool ShowJoints { get; set; }
        public bool ShowNames { get; set; }
        public enum FingerModes
        {
            Off,
            Synchronous,
            Asynchronous
        };
        [Setting]
        public FingerModes FingerMode;
        [Setting]
        public bool AllowInferredContacts;
        [Setting]
        public string Source;
        private AutoResetEvent[] ResetEvents = { new
AutoResetEvent(false), new AutoResetEvent(false) };
        private FingerFinder LeftFingerFinder;
        private FingerFinder RightFingerFinder;
        private long ContactID = 0;
        public TuioContact ThumbLeft;
        public TuioContact IndexFingerLeft;
        public TuioContact MiddleFingerLeft;
        public TuioContact RingFingerLeft;
        public TuioContact PinkyLeft;
        public TuioContact ThumbRight;
```

```
        public TuioContact IndexFingerRight;
        public TuioContact MiddleFingerRight;
        public TuioContact RingFingerRight;
        public TuioContact PinkyRight;
        public TuioContact HipCenter;
        public TuioContact Spine;
        public TuioContact ShoulderCenter;
        public TuioContact Head;
        public TuioContact ShoulderLeft;
        public TuioContact ElbowLeft;
        public TuioContact WristLeft;
        public TuioContact HandLeft;
        public TuioContact ShoulderRight;
        public TuioContact ElbowRight;
        public TuioContact WristRight;
        public TuioContact HandRight;
        public TuioContact HipLeft;
        public TuioContact KneeLeft;
        public TuioContact AnkleLeft;
        public TuioContact FootLeft;
        public TuioContact HipRight;
        public TuioContact KneeRight;
        public TuioContact AnkleRight;
        public TuioContact FootRight;
        private  List<TuioContact>  contacts  =  new
List<TuioContact>(30);
        private              List<Action<TuioContact>>
contactAddedListeners              =              new
List<Action<TuioContact>>();
        private              List<Action<TuioContact>>
contactChangedListeners              =              new
List<Action<TuioContact>>();
        private              List<Action<TuioContact>>
contactRemovedListeners              =              new
List<Action<TuioContact>>();
        private HandDataSourceSettings handSettings;
        private IHandDataSource handDataSource;
        private HandData leftHand;
        private HandData rightHand;
        public List<TuioContact> getContacts()
        {
            return new List<TuioContact>(contacts);
        }
```

```
        public                                          void
registerContactAddedListener(Action<TuioContact>
listener)
        {
            contactAddedListeners.Add(listener);
        }
        public                                          void
registerContactChangedListener(Action<TuioContact>
listener)
        {
            contactChangedListeners.Add(listener);
        }
        public                                          void
registerContactRemovedListener(Action<TuioContact>
listener)
        {
            contactRemovedListeners.Add(listener);
        }
            : this(new KinectSensor, new Canvas, new
KinectSettings(), new FingerSettings())
        {
        }

        public KinectAdapter(KinectSettings settings)
            : this(settings, new FingerSettings())
        {
        }

        public KinectAdapter(KinectSensor rootSensor,
Canvas rootCanvas, KinectSettings kinectSettings,
FingerSettings fingerSettings, HandDataSourceSettings
handSettingsRoot, IHandDataSource handDataSourceRoot )
        {
            this.sensor = rootSensor;
            this.canvas = rootCanvas;
            this.handSettings = handSettingsRoot;
            this.handDataSource = handDataSourceRoot;

            FingerMode = kinectSettings.FingerMode;
            AllowInferredContacts                       =
kinectSettings.AllowInferredContacts;
            Source = kinectSettings.Source;
```

```
            contacts.Add(ThumbLeft          =          new
TuioContact(Source,   ContactID++,   0,   0,   0,   0,
TuioContactType.ThumbLeft));
            contacts.Add(IndexFingerLeft      =      new
TuioContact(Source,   ContactID++,   1,   0,   0,   0,
TuioContactType.IndexFingerLeft));
            contacts.Add(MiddleFingerLeft     =      new
TuioContact(Source,   ContactID++,   2,   0,   0,   0,
TuioContactType.MiddleFingerLeft));
            contacts.Add(RingFingerLeft       =      new
TuioContact(Source,   ContactID++,   3,   0,   0,   0,
TuioContactType.RingFingerLeft));
            contacts.Add(PinkyLeft          =          new
TuioContact(Source,   ContactID++,   4,   0,   0,   0,
TuioContactType.PinkyLeft));
            contacts.Add(ThumbRight         =          new
TuioContact(Source,   ContactID++,   5,   0,   0,   0,
TuioContactType.ThumbRight));
            contacts.Add(IndexFingerRight     =      new
TuioContact(Source,   ContactID++,   6,   0,   0,   0,
TuioContactType.IndexFingerRight));
            contacts.Add(MiddleFingerRight    =      new
TuioContact(Source,   ContactID++,   7,   0,   0,   0,
TuioContactType.MiddleFingerRight));
            contacts.Add(RingFingerRight      =      new
TuioContact(Source,   ContactID++,   8,   0,   0,   0,
TuioContactType.RingFingerRight));
            contacts.Add(PinkyRight         =          new
TuioContact(Source,   ContactID++,   9,   0,   0,   0,
TuioContactType.PinkyRight));
            contacts.Add(HipCenter          =          new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.HipCenter));
            contacts.Add(Spine            =            new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.Spine));
            contacts.Add(ShoulderCenter       =      new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.ShoulderCenter));
            contacts.Add(Head             =            new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.Head));
            contacts.Add(ShoulderLeft       =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.ShoulderLeft));
```

```
            contacts.Add(ElbowLeft        =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.ElbowLeft));
            contacts.Add(WristLeft        =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.WristLeft));
            contacts.Add(HandLeft         =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.HandLeft));
            contacts.Add(ShoulderRight      =      new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.ShoulderRight));
            contacts.Add(ElbowRight       =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.ElbowRight));
            contacts.Add(WristRight       =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.WristRight));
            contacts.Add(HandRight        =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.HandRight));
            contacts.Add(HipLeft         =         new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.HipLeft));
            contacts.Add(KneeLeft         =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.KneeLeft));
            contacts.Add(AnkleLeft        =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.AnkleLeft));
            contacts.Add(FootLeft         =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.FootLeft));
            contacts.Add(HipRight         =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.HipRight));
            contacts.Add(KneeRight        =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.KneeRight));
            contacts.Add(AnkleRight       =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.AnkleRight));
            contacts.Add(FootRight        =        new
TuioContact(Source,   ContactID++,   -1,   0,   0,   0,
TuioContactType.FootRight));
```

```
            if (KinectSensor.KinectSensors.Count > 0)
            {
                this.sensor                              =
KinectSensor.KinectSensors.Where(item  =>  item.Status
== KinectStatus.Connected).FirstOrDefault();
            }
            else
            {
                System.Windows.MessageBox.Show("No
Device Connected");

System.Windows.Application.Current.Shutdown();
                return;
            }

this.sensor.DepthStream.Enable(DepthImageFormat.Resolu
tion640x480Fps30);
            this.sensor.DepthFrameReady             +=
this.KinectDepthFrameReady;
            this.sensor.SkeletonFrameReady   +=   new
EventHandler<SkeletonFrameReadyEventArgs>(KinectSkelet
onFrameReady);
            this.sensor.Start();

            //if (HandLeft.active)
            //{
            LeftFingerFinder           =           new
FingerFinder(ResetEvents[0],                    this,
FingerFinder.HandType.Left,            fingerSettings,
handSettings, handDataSource);
            //}

            //if (HandRight.active)
            //{
            RightFingerFinder          =           new
FingerFinder(ResetEvents[1],                    this,
FingerFinder.HandType.Right,           fingerSettings,
handSettings, handDataSource);
            //}
        }

        private   void   KinectDepthFrameReady(object
sender, DepthImageFrameReadyEventArgs e)
        {
```

```
            using (DepthImageFrame depthFrame =
e.OpenDepthImageFrame())
            {
                if (depthFrame != null)
                {
                    // Copy the pixel data from the
image to a temporary array
                    this.depthPixelData = new
short[depthFrame.PixelDataLength];

depthFrame.CopyPixelDataTo(this.depthPixelData);

                    this.depthFrameWidth =
depthFrame.Width;
                    this.DepthFrameData = depthFrame;

                    // Get the min and max reliable
depth for the current frame
                    int minDepth =
depthFrame.MinDepth;
                    int maxDepth =
depthFrame.MaxDepth;
                }
            }
        }
        void KinectSkeletonFrameReady(object sender,
SkeletonFrameReadyEventArgs e)
        {
            canvas.Children.Clear();
            Skeleton[] skeletons = null;
            JointCollection joints;

            using (var frame = e.OpenSkeletonFrame())
            {
                if (frame != null)
                {
                    skeletons = new
Skeleton[frame.SkeletonArrayLength];

frame.CopySkeletonDataTo(skeletons);
                }
            }
            if (skeletons == null) return;
```

```
            skeleton = (from trackSkeleton in
skeletons
                        where
trackSkeleton.TrackingState                          ==
SkeletonTrackingState.Tracked
                        select
trackSkeleton).FirstOrDefault();

            if (skeleton == null) return;

            foreach (var data in skeletons)
                if (SkeletonTrackingState.Tracked ==
data.TrackingState)
                {
                    joints = data.Joints;

getJointPosition(joints[JointType.HandLeft],
HandLeft);

                    if (FingerMode != FingerModes.Off
&& HandLeft.active)

//ThreadPool.QueueUserWorkItem(LeftFingerFinder.finger
sReady, ResetEvents[0]);


getJointPosition(joints[JointType.HandRight],
HandRight);

                    if (FingerMode != FingerModes.Off
&& HandRight.active)

//ThreadPool.QueueUserWorkItem(RightFingerFinder.proce
ssHand, ResetEvents[1]);

                    if          (FingerMode          ==
FingerModes.Synchronous)
                    {
                        ResetEvents[0].WaitOne(100);
                        ResetEvents[1].WaitOne(100);
                    }

getJointPosition(joints[JointType.HipCenter],
HipCenter);
```

```
getJointPosition(joints[JointType.Spine], Spine);

getJointPosition(joints[JointType.ShoulderCenter],
ShoulderCenter);
getJointPosition(joints[JointType.Head],        Head);
getJointPosition(joints[JointType.ShoulderLeft],
ShoulderLeft);
getJointPosition(joints[JointType.ElbowLeft],
ElbowLeft);
getJointPosition(joints[JointType.WristLeft],
WristLeft);
getJointPosition(joints[JointType.ShoulderRight],
ShoulderRight);
getJointPosition(joints[JointType.ElbowRight],
ElbowRight);
getJointPosition(joints[JointType.WristRight],
WristRight);
getJointPosition(joints[JointType.HipLeft],  HipLeft);
getJointPosition(joints[JointType.KneeLeft],
KneeLeft);
getJointPosition(joints[JointType.AnkleLeft],
AnkleLeft);
getJointPosition(joints[JointType.FootLeft],
FootLeft);
getJointPosition(joints[JointType.HipRight],
HipRight);
getJointPosition(joints[JointType.KneeRight],
KneeRight);
getJointPosition(joints[JointType.AnkleRight],
AnkleRight);
getJointPosition(joints[JointType.FootRight],
FootRight);
                }
            if
(this.sensor.SkeletonStream.TrackingMode            ==
SkeletonTrackingMode.Seated)
            {
                ShowJoints = true;
                DrawSeatedSkeleton();
            }
            else
            {
                ShowJoints = true;
                DrawDefaultSkeleton();
```

```
            }
        }

        public short[] getDepthFrame()
        {
            return depthPixelData;
        }

        public void shutdown()
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
            Environment.Exit(0);
            this.StopTracking();
        }

        public void StopTracking()
        {
            if     (this.sensor     !=     null     &&
this.sensor.SkeletonStream.IsEnabled)
            {
                this.sensor.SkeletonStream.Disable();
                this.sensor.SkeletonFrameReady -= new
EventHandler<SkeletonFrameReadyEventArgs>(KinectSkelet
onFrameReady);
            }
            canvas.Children.Clear();
        }

        public void StartTracking()
        {
            TransformSmoothParameters smoothParameters
= new TransformSmoothParameters
            {
                Correction = 0.1f,
                JitterRadius = 0.05f,
                MaxDeviationRadius = 0.05f,
                Prediction = 0.5f,
                Smoothing = 0.5f
            };

            if (this.sensor != null)
            {
```

```
this.sensor.SkeletonStream.Enable(smoothParameters);
this.sensor.SkeletonStream.TrackingMode              =
SkeletonTrackingMode.Seated;
                this.sensor.SkeletonFrameReady += new
EventHandler<SkeletonFrameReadyEventArgs>(KinectSkelet
onFrameReady);
            }
        }
        public                                      void
runContactAddedListeners(TuioContact contact)
        {
            foreach (Action<TuioContact> listener in
contactAddedListeners)
                listener(contact);
        }
        public                                      void
runContactChangedListeners(TuioContact contact)
        {
            foreach (Action<TuioContact> listener in
contactChangedListeners)
                listener(contact);
        }
        public                                      void
runContactRemovedListeners(TuioContact contact)
        {
            foreach (Action<TuioContact> listener in
contactRemovedListeners)
                listener(contact);
        }
        public  void  getJointPosition(Joint  joint,
TuioContact contact)
        {
            if         (joint.TrackingState         ==
JointTrackingState.Tracked  ||  (AllowInferredContacts
&&               joint.TrackingState               ==
JointTrackingState.Inferred))
            {
                DepthImagePoint      depthPoint      =
sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(j
oint.Position,
DepthImageFormat.Resolution640x480Fps30);

                int pixelIndex = (int)(depthPoint.X +
((int) depthPoint.Y * this.depthFrameWidth));
```

```
                realDepth                                   =
this.depthPixelData[pixelIndex]                            >>
DepthImageFrame.PlayerIndexBitmaskWidth;
                if (realDepth == 0)
                {
                    if (contact.active)
                    {
                        contact.active    =    false;
runContactRemovedListeners(contact);
                    }
                    return;
                }
                contact.xpos = depthPoint.X;
                contact.ypos = depthPoint.Y;
                contact.zpos = realDepth;
                if (!contact.active)
                {
                    contact.active = true;
                    contact.session_id = ContactID++;
                    runContactAddedListeners(contact);
                }
                else
runContactChangedListeners(contact);
            }
            else if (contact.active)
            {
                contact.active = false;
                runContactRemovedListeners(contact);
            }
        }
        void   drawBone(Joint   trackedJoint1,   Joint
trackedJoint2)
        {
            Line bone = new Line();
            bone.Stroke = Brushes.Red;
            bone.StrokeThickness = 3;
            Point               joint1              =
this.ScalePosition(trackedJoint1.Position);
            bone.X1 = joint1.X;
            bone.Y1 = joint1.Y;
            if (this.ShowNames)
            {
                Point          mappedPoint1          =
this.ScalePosition(trackedJoint1.Position);
```

```
                TextBlock depthText = new TextBlock();
                //getJointPosition(trackedJoint1);
                depthText.Text = realDepth.ToString();
                depthText.Foreground                    =
Brushes.DeepPink;
                Canvas.SetLeft(depthText,
mappedPoint1.X + 15);
                Canvas.SetTop(depthText,
mappedPoint1.Y + 15);
                canvas.Children.Add(depthText);

                TextBlock textBlock = new TextBlock();
                textBlock.Text                          =
trackedJoint1.JointType.ToString();
                textBlock.Foreground = Brushes.Yellow;
                Canvas.SetLeft(textBlock,
mappedPoint1.X + 5);
                Canvas.SetTop(textBlock,
mappedPoint1.Y + 5);
                canvas.Children.Add(textBlock);
            }

            if (this.ShowJoints)
            {
                Point          mappedPoint1          =
this.ScalePosition(trackedJoint1.Position);
                Rectangle   r   =   new   Rectangle();
r.Height = 10; r.Width = 10;
                r.Fill = Brushes.Red;
                Canvas.SetLeft(r, mappedPoint1.X - 2);
                Canvas.SetTop(r, mappedPoint1.Y - 2);
                canvas.Children.Add(r);
            }
            Point               joint2               =
this.ScalePosition(trackedJoint2.Position);
            bone.X2 = joint2.X;
            bone.Y2 = joint2.Y;
            Point            mappedPoint2            =
this.ScalePosition(trackedJoint2.Position);

            if       (LeafJoint(trackedJoint2)       &&
this.ShowJoints)
            {
                Rectangle   r1   =   new   Rectangle();
r1.Height = 10; r1.Width = 10;
```

```
                r1.Fill = Brushes.Red;
                Canvas.SetLeft(r1,  mappedPoint2.X  -
2);
                Canvas.SetTop(r1, mappedPoint2.Y - 2);
                canvas.Children.Add(r1);
            }
            if      (LeafJoint(trackedJoint2)      &&
this.ShowJoints)
            {
                Point          mappedPoint          =
this.ScalePosition(trackedJoint2.Position);
                TextBlock textBlock = new TextBlock();
                textBlock.Text                       =
trackedJoint2.JointType.ToString();
                textBlock.Foreground = Brushes.Yellow;
                Canvas.SetLeft(textBlock,
mappedPoint.X + 5);
                Canvas.SetTop(textBlock, mappedPoint.Y
+ 5);
                canvas.Children.Add(textBlock);
            }
            canvas.Children.Add(bone);
        }
        private bool LeafJoint(Joint j2)
        {
            if (j2.JointType == JointType.HandRight ||
j2.JointType == JointType.HandLeft || j2.JointType ==
JointType.FootLeft      ||      j2.JointType      ==
JointType.FootRight)
            {
                return true;
            }
            return false;
        }
        private   Point   ScalePosition(SkeletonPoint
skeletonPoint)
        {
            DepthImagePoint        depthPoint        =
this.sensor.CoordinateMapper.MapSkeletonPointToDepthPo
int(skeletonPoint,
DepthImageFormat.Resolution640x480Fps30);
            return       new       Point(depthPoint.X,
depthPoint.Y);
        }
```

```
        public void DrawDefaultSkeleton()
        {
            DrawSkeleton();
        }
        public void DrawSeatedSkeleton()
        {
            DrawHeadShoulder();
            DrawLeftArm();
            DrawRightArm();
        }
        public void DrawHeadShoulder()
        {
            drawBone(skeleton.Joints[JointType.Head],
skeleton.Joints[JointType.ShoulderCenter]);
        }
        public void DrawSpine()
        {
            drawBone(skeleton.Joints[JointType.Head],
skeleton.Joints[JointType.ShoulderCenter]);

drawBone(skeleton.Joints[JointType.ShoulderCenter],
skeleton.Joints[JointType.Spine]);
        }
        public void DrawLeftArm()
        {

drawBone(skeleton.Joints[JointType.ShoulderCenter],
skeleton.Joints[JointType.ShoulderLeft]);

drawBone(skeleton.Joints[JointType.ShoulderLeft],
skeleton.Joints[JointType.ElbowLeft]);

drawBone(skeleton.Joints[JointType.ElbowLeft],
skeleton.Joints[JointType.WristLeft]);

drawBone(skeleton.Joints[JointType.WristLeft],
skeleton.Joints[JointType.HandLeft]);
        }
        public void DrawRightArm()
        {

drawBone(skeleton.Joints[JointType.ShoulderCenter],
skeleton.Joints[JointType.ShoulderRight]);
```

```
drawBone(skeleton.Joints[JointType.ShoulderRight],
skeleton.Joints[JointType.ElbowRight]);
drawBone(skeleton.Joints[JointType.ElbowRight],
skeleton.Joints[JointType.WristRight]);
drawBone(skeleton.Joints[JointType.WristRight],
skeleton.Joints[JointType.HandRight]);
        }
        public void DrawSkeleton()
        {
            drawBone(skeleton.Joints[JointType.Head],
skeleton.Joints[JointType.ShoulderCenter]);
drawBone(skeleton.Joints[JointType.ShoulderCenter],
skeleton.Joints[JointType.Spine]);
drawBone(skeleton.Joints[JointType.ShoulderCenter],
skeleton.Joints[JointType.ShoulderLeft]);
drawBone(skeleton.Joints[JointType.ShoulderLeft],
skeleton.Joints[JointType.ElbowLeft]);
drawBone(skeleton.Joints[JointType.ElbowLeft],
skeleton.Joints[JointType.WristLeft]);
drawBone(skeleton.Joints[JointType.WristLeft],
skeleton.Joints[JointType.HandLeft]);
drawBone(skeleton.Joints[JointType.ShoulderCenter],
skeleton.Joints[JointType.ShoulderRight]);
drawBone(skeleton.Joints[JointType.ShoulderRight],
skeleton.Joints[JointType.ElbowRight]);
drawBone(skeleton.Joints[JointType.ElbowRight],
skeleton.Joints[JointType.WristRight]);
drawBone(skeleton.Joints[JointType.WristRight],
skeleton.Joints[JointType.HandRight]);
drawBone(skeleton.Joints[JointType.Spine],
skeleton.Joints[JointType.HipCenter]);
drawBone(skeleton.Joints[JointType.HipCenter],
skeleton.Joints[JointType.HipLeft]);
drawBone(skeleton.Joints[JointType.HipLeft],
skeleton.Joints[JointType.KneeLeft]);
drawBone(skeleton.Joints[JointType.KneeLeft],
skeleton.Joints[JointType.AnkleLeft]);
drawBone(skeleton.Joints[JointType.AnkleLeft],
skeleton.Joints[JointType.FootLeft]);
drawBone(skeleton.Joints[JointType.HipCenter],
skeleton.Joints[JointType.HipRight]);
drawBone(skeleton.Joints[JointType.HipRight],
skeleton.Joints[JointType.KneeRight]);
```

```
drawBone(skeleton.Joints[JointType.KneeRight],
skeleton.Joints[JointType.AnkleRight]);
drawBone(skeleton.Joints[JointType.AnkleRight],
skeleton.Joints[JointType.FootRight]);
        }
    }
}
```

C.5 Kode File KinectSettings.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace GRIP
{
    public class KinectSettings : ISettingManager
    {
        public KinectAdapter.FingerModes FingerMode =
KinectAdapter.FingerModes.Synchronous;
        public bool AllowInferredContacts = true;
        public string Source;
    }
}
```

C.6 Kode File CountHistory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace VisualTester
{
    public class CountHistory
    {
        private Queue<int> queue;
        public CountHistory()
        {
            this.queue = new Queue<int>();
        }
        public void Add(int value)
        {
            this.queue.Enqueue(value);
            if (this.queue.Count > this.Length)
```

```
            {
                this.queue.Dequeue();
            }
        }
        public int Length { get; set; }
        public bool AllEqual()
        {
            if (this.queue.Count <= 1)
            {
                return true;
            }
            var value = this.queue.First();
            return this.queue.All(v => v == value);
        }
    }
}
```

C.7 Kode File HandInterfaceElement.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Controls;
using System.Windows.Shapes;
using System.Windows.Media;
using System.Windows;
using System.Globalization;
using System.Windows.Media.Animation;
using System.Timers;
using CCT.NUI.HandTracking;
using CCT.NUI.Core;
using CCT.NUI.Core.Video;
using CCT.NUI.Visual;
using CCT.NUI.KinectSDK;
using CCT.NUI.Core.Clustering;
using System.Windows.Threading;

namespace VisualTester
{
    public class HandInterfaceElement : UIElement
    {
        private HandCollection currentData = new
HandCollection();
        private Timer timer;
```

```
        private CountHistory countHistory;
        private static bool stopped = false;
        private int fingerPointCount = 0;
        private InterfacePainter painter;
        public event EventHandler Reset;
        public event ElapsedEventHandler CharEnter;
        public HandInterfaceElement()
        {
            this.timer = new Timer();
            this.timer.Interval = 1000;
            this.timer.Elapsed         +=         new
ElapsedEventHandler(timer_Elapsed);
            this.painter = new InterfacePainter();
            this.countHistory  =  new  CountHistory  {
Length = 3 };
            this.Value = 0;
        }
        private            void            FadeIn(int
durationInMilliseconds)
        {
            var animation = new ByteAnimation(100, new
Duration(TimeSpan.FromMilliseconds(durationInMilliseco
nds)));
this.BeginAnimation(HandInterfaceElement.InterfaceOpac
ityProperty, animation);
        }
        private            void            FadeOut(int
durationInMilliseconds)
        {
            this.timer.Enabled = false;
            var animation = new ByteAnimation(0, new
Duration(TimeSpan.FromMilliseconds(durationInMilliseco
nds)));
this.BeginAnimation(HandInterfaceElement.InterfaceOpac
ityProperty, animation);
        }
        public byte InterfaceOpacity
        {
            get                {                return
(byte)GetValue(InterfaceOpacityProperty); }
            set  {  SetValue(InterfaceOpacityProperty,
value); }
        }
        public  static  readonly  DependencyProperty
InterfaceOpacityProperty                              =
```

```
DependencyProperty.Register("InterfaceOpacity",
typeof(byte),     typeof(HandInterfaceElement),     new
PropertyMetadata(OnDependencyPropertyChanged));
        public double Value
        {
            get                 {                 return
(double)GetValue(ValueProperty); }
            set { SetValue(ValueProperty, value); }
        }
        public   static   readonly   DependencyProperty
ValueProperty  =  DependencyProperty.Register("Value",
typeof(double),    typeof(HandInterfaceElement),    new
PropertyMetadata(OnDependencyPropertyChanged));
        protected             override             void
OnRender(DrawingContext drawingContext)
        {
            base.OnRender(drawingContext);
            if (this.currentData.Count > 0)
            {
this.painter.DrawHand(this.currentData.Hands.First(),
this.currentData, drawingContext);
            }
        }
        private               static               void
OnDependencyPropertyChanged(DependencyObject        d,
DependencyPropertyChangedEventArgs e)
        {
            (d                                          as
HandInterfaceElement).InvalidateVisual();
        }
        public int Number
        {
            get { return this.fingerPointCount; }
        }
        public void ResetText()
        {
            if (this.Reset != null)
            {
                this.Reset(this, EventArgs.Empty);
            }
        }
        private         void         StartAnimation(int
durationInMilliseconds)
        {
             this.FadeIn(500);
```

```
            this.timer.Enabled = true;
        }
        void        timer_Elapsed(object        sender,
ElapsedEventArgs e)
        {
            if (this.CharEnter != null)
            {
                this.CharEnter(this, e);
            }
        }
        internal       void       Update(HandCollection
handCollection)
        {
            if (stopped)
            {
                return;
            }
            this.Dispatcher.Invoke(new Action(() =>
                {
                    this.currentData = handCollection;
                    if (!handCollection.IsEmpty)
                    {
                        var           hand           =
handCollection.Hands.First();
                        this.UpdateFingerCount(hand);
                        UpdateHandValues(hand);
                        this.StartAnimation(1000);
                    }
                    else  if  ((handCollection.IsEmpty)
&& this.InterfaceOpacity == 100)
                    {
                        this.FadeOut(500);
                    }
                    InvalidateVisual();
                }));
        }
        private void UpdateFingerCount(HandData hand)
        {
            this.countHistory.Add(hand.FingerCount);
            if (this.countHistory.AllEqual())
            {
                this.fingerPointCount = hand.FingerCo
unt;
            }
        }
```

```
        private void UpdateHandValues(HandData hand)
        {
            if (fingerPointCount >= 5)
            {
                this.StartAnimation(1000);
            }
            else if (hand.HasFingers == false)
            {
                this.timer.Enabled = false;
            }
            else if (fingerPointCount > 0)
            {
                this.timer.Enabled = true;
            }
        }
    }
}
```

C.8 Kode File InterfacePainter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows;
using System.Globalization;
using CCT.NUI.Touch;
using CCT.NUI.HandTracking;

namespace VisualTester
{
    public class InterfacePainter
    {
        private FormattedText number;
        private Typeface typeFace = new Typeface(new
FontFamily("Arial"),                FontStyles.Normal,
FontWeights.Bold, FontStretches.Normal);
        public InterfacePainter ()
           {
           }
        private Point ConvertRadianToCartesian(double
angle, double radius)
        {
```

```
            const double FULL_ARC = 360;
            var angleRadius = (Math.PI / (FULL_ARC /
2)) * (angle - FULL_ARC / 4);
            var x = radius * Math.Cos(angleRadius);
            var y = radius * Math.Sin(angleRadius);
            return new Point(x, y);
        }
        internal   void   DrawHand(HandData   handData,
HandCollection     handCollection,     DrawingContext
drawingContext)
        {
                this.WriteNumberText(handData,
handCollection, drawingContext);
                this.DrawFingerPoints(handCollection,
drawingContext);
        }

        protected             virtual             void
WriteNumberText(HandData   handData,   HandCollection
handCollection, DrawingContext drawingContext)
        {
          foreach (var hand in handCollection.Hands)
          {
            foreach (var point in hand.FingerPoints)
            {
                var         brush         =         new
SolidColorBrush(Color.FromRgb(0, 180, 255));

this.CreateNumberText(point.Id.ToString(),      brush,
drawingContext, point);
            }
          }
        }
        private  void  CreateNumberText(string  text,
Brush    brush,    DrawingContext    drawingContext,
FingerPoint point)
        {
            this.number   =   new   FormattedText(text,
CultureInfo.CurrentCulture, FlowDirection.LeftToRight,
this.typeFace, 16, brush);
            drawingContext.DrawText(this.number,   new
Point(point.X, point.Y));
        }
```

```
       protected              virtual              void
DrawFingerPoints(HandCollection                cluster,
DrawingContext drawingContext)
        {
            foreach (var hand in cluster.Hands)
            {
                foreach       (var       point       in
hand.FingerPoints)
                {
                    PaintFingerPoint(point,
drawingContext);
                }
            }
        }
        private    void    PaintFingerPoint(FingerPoint
point, DrawingContext drawingContext)
        {
            drawingContext.DrawEllipse(new
SolidColorBrush(Color.FromRgb(255,  0,  0)),  null,  new
Point(point.X, point.Y), 5.5, 5.5);
        }
    }
}
```

C.9 Kode File MainWindow.xaml.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Timers;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Media.Media3D;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Runtime.InteropServices;
using CCT.NUI.Core;
using CCT.NUI.Core.Video;
using CCT.NUI.Visual;
```

```
using CCT.NUI.HandTracking;
using CCT.NUI.KinectSDK;
using CCT.NUI.Core.Clustering;
using System.Windows.Forms;
using Microsoft.Kinect;
using System.Threading;
using TUIO;
using GRIP;

namespace VisualTester
{
    public partial class MainWindow : Window
    {
        private KinectSensor sensor;
        private Skeleton skeleton;
        private KinectAdapter kinectAdapter;
        private bool skeltonNames = false;
        System.Windows.Forms.Timer timer;
        System.Windows.Forms.Timer timerShow;
        System.Windows.Forms.Timer timerData;
        byte[] depthFrame32 = new byte[640 * 480 * 4];
        private TuioContact[][] kinect_points;
        private     Brush     blueBrush     =     new
SolidColorBrush(Color.FromRgb(0, 0, 255));
        private     Brush     greenBrush     =     new
SolidColorBrush(Color.FromRgb(0, 255, 0));
        private     Brush     redBrush     =     new
SolidColorBrush(Color.FromRgb(255, 0, 0));
        private     Brush     yellowBrush     =     new
SolidColorBrush(Color.FromRgb(255, 255, 0));
        public    Brush    SkeletonBrush    =    new
SolidColorBrush(Colors.Green);
        private List<TuioContact> fingersLeft;
        private List<TuioContact> fingersRight;
        private   Brush[]   fingerBrushes   =   {   new
SolidColorBrush(Color.FromRgb(0, 255, 0)),
                                                  new
SolidColorBrush(Color.FromRgb(155, 255, 0)),
                                                  new
SolidColorBrush(Color.FromRgb(255, 255, 0)),
                                                  new
SolidColorBrush(Color.FromRgb(255, 255, 155)),
                                                  new
SolidColorBrush(Color.FromRgb(255, 255, 255))
                                                 };
```

```
        private List<TextBlock> labelsLeft = new
List<TextBlock>();
        private List<TextBlock> labelsRight = new
List<TextBlock>();
        private IDataSourceFactory factory;
        private IHandDataSource handDataSource;
        private IClusterDataSource clusterDataSource;
        private IImageDataSource rgbImageDataSource;
        private HandCollection currentData = new
HandCollection();
        private FingerFinder LeftFingerFinder;
        private FingerFinder RightFingerFinder;
        private string gestureMeaning;
        int ctr = 0;
        private CCT.NUI.Core.Point handPalm = new
CCT.NUI.Core.Point(0, 0, 0);
        private List<TuioContact> fingerPointsDraw;
        private TuioContact handDraw;
        private List<TextBlock> labelsDraw;
        private bool handChooseThread = false;
        private bool gestureChooseThread = false;
        private bool handType = true;
        private TuioContact Wrist, Hand, Thumb,
IndexFinger, MiddleFinger, RingFinger, Pinky;
        private double sudutTangan;
        private int clusterPointCount;
        private double AngleScalingFactor;
        private HandInterfaceElement interfaceElement;
        public MainWindow()
        {
            InitializeComponent();
            this.initialize();
        }
        private void initialize()
        {
            this.Cursor                                =
System.Windows.Input.Cursors.Wait;

            this.factory             =             new
SDKDataSourceFactory(useNearMode: false);
            this.clusterDataSource                     =
this.factory.CreateClusterDataSource(new
ClusterDataSourceSettings { MaximumDepthThreshold =
1000 });
```

```
            this.handDataSource          =          new
HandDataSource(this.factory.CreateShapeDataSource(this
.clusterDataSource,                                    new
CCT.NUI.Core.Shape.ShapeDataSourceSettings()));
            this.rgbImageDataSource                    =
this.factory.CreateRGBImageDataSource();
            this.rgbImageDataSource.Start();

            var           depthImageSource             =
this.factory.CreateDepthImageDataSource();
            depthImageSource.NewDataAvailable  +=  new
NewDataHandler<ImageSource>(MainWindow_NewDataAvailabl
e);
            depthImageSource.Start();

            this.interfaceElement         =         new
HandInterfaceElement();
            interfaceElement.CharEnter     +=      new
System.Timers.ElapsedEventHandler(element_Tick);
            interfaceElement.Reset        +=        new
EventHandler(element_Reset);

            this.handDataSource.NewDataAvailable    +=
new
NewDataHandler<HandCollection>(handDataSource_NewDataA
vailable);
            handDataSource.Start();
            this.Cursor                                =
System.Windows.Input.Cursors.Arrow;
        }
        private   void   Window_Loaded(object   sender,
EventArgs e)
        {
            ConfigParser       parser       =       new
ConfigParser("config.xml");
            KinectSettings   kinectSettings   =   new
KinectSettings();
            FingerSettings   fingerSettings   =   new
FingerSettings();
            try
            {

parser.LoadConfiguration(kinectSettings);

parser.LoadConfiguration(fingerSettings);
```

```
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
            HandDataSourceSettings handSettings = new
HandDataSourceSettings();
            kinectAdapter = new KinectAdapter(sensor,
canvSkeleton,      kinectSettings,      fingerSettings,
handSettings, handDataSource);
            kinectAdapter.StartTracking();
            var layers = new List<IWpfLayer>();
            layers.Add(new
WpfHandLayer(this.handDataSource));
            layers.Add(new
WpfClusterLayer(this.clusterDataSource));
            this.videoControl.Layers = layers;
            kinect_points = new TuioContact[][]
            {
                new          TuioContact[]          {
kinectAdapter.Head,      kinectAdapter.ShoulderCenter,
kinectAdapter.Spine, kinectAdapter.HipCenter },
                new          TuioContact[]          {
kinectAdapter.ShoulderCenter,
kinectAdapter.ShoulderLeft,   kinectAdapter.ElbowLeft,
kinectAdapter.WristLeft, kinectAdapter.HandLeft },
                new          TuioContact[]          {
kinectAdapter.ShoulderCenter,
kinectAdapter.ShoulderRight, kinectAdapter.ElbowRight,
kinectAdapter.WristRight, kinectAdapter.HandRight },
                new          TuioContact[]          {
kinectAdapter.HipCenter,        kinectAdapter.HipLeft,
kinectAdapter.KneeLeft,       kinectAdapter.AnkleLeft,
kinectAdapter.FootLeft },
                new          TuioContact[]          {
kinectAdapter.HipCenter,       kinectAdapter.HipRight,
kinectAdapter.KneeRight,     kinectAdapter.AnkleRight,
kinectAdapter.FootRight }
            };
            fingersLeft  =  new  List<TuioContact>(new
TuioContact[]        {        kinectAdapter.ThumbLeft,
kinectAdapter.IndexFingerLeft,
kinectAdapter.MiddleFingerLeft,
kinectAdapter.RingFingerLeft,  kinectAdapter.PinkyLeft
});
```

```
            fingersRight = new List<TuioContact>(new
TuioContact[]        {        kinectAdapter.ThumbRight,
kinectAdapter.IndexFingerRight,
kinectAdapter.MiddleFingerRight,
kinectAdapter.RingFingerRight,
kinectAdapter.PinkyRight });
            {
                TextBlock     thumbLabel     =     new
TextBlock();
                thumbLabel.Foreground = yellowBrush;
                thumbLabel.Text = "Thumb";
                TextBlock     indexLabel     =     new
TextBlock();
                indexLabel.Foreground = yellowBrush;
                indexLabel.Text = "Index";
                TextBlock    middleLabel     =     new
TextBlock();
                middleLabel.Foreground = yellowBrush;
                middleLabel.Text = "Middle";
                TextBlock ringLabel = new TextBlock();
                ringLabel.Foreground = yellowBrush;
                ringLabel.Text = "Ring";
                TextBlock     pinkyLabel     =     new
TextBlock();
                pinkyLabel.Foreground = yellowBrush;
                pinkyLabel.Text = "Pinky";
                labelsLeft.Add(thumbLabel);
                labelsLeft.Add(indexLabel);
                labelsLeft.Add(middleLabel);
                labelsLeft.Add(ringLabel);
                labelsLeft.Add(pinkyLabel);
            }
            {
                TextBlock     thumbLabel     =     new
TextBlock();
                thumbLabel.Foreground = yellowBrush;
                thumbLabel.Text = "Thumb";
                TextBlock     indexLabel     =     new
TextBlock();
                indexLabel.Foreground = yellowBrush;
                indexLabel.Text = "Index";
                TextBlock    middleLabel     =     new
TextBlock();
                middleLabel.Foreground = yellowBrush;
                middleLabel.Text = "Middle";
```

```
                TextBlock ringLabel = new TextBlock();
                ringLabel.Foreground = yellowBrush;
                ringLabel.Text = "Ring";
                TextBlock     pinkyLabel     =     new
TextBlock();
                pinkyLabel.Foreground = yellowBrush;
                pinkyLabel.Text = "Pinky";
                labelsRight.Add(thumbLabel);
                labelsRight.Add(indexLabel);
                labelsRight.Add(middleLabel);
                labelsRight.Add(ringLabel);
                labelsRight.Add(pinkyLabel);
            }
            this.fingerPointsDraw = fingersRight;
            this.handDraw = kinectAdapter.HandRight;
            this.labelsDraw = labelsRight;
            handChooseThread = true;
            timer = new System.Windows.Forms.Timer();
            timer.Interval = 40;
            timer.Enabled = true;
            timer.Tick             +=             new
EventHandler(timer_Elapsed);
            timer.Start();
            timerShow              =              new
System.Windows.Forms.Timer();
            timerShow.Interval = 1000;
            timerShow.Enabled = true;
            timerShow.Tick           +=            new
EventHandler(timer_Show);
            timerShow.Start();
            timerData              =              new
System.Windows.Forms.Timer();
            timerData.Interval = 500;
            timerData.Enabled = true;
            timerData.Tick           +=            new
EventHandler(timer_Data);
            timerData.Start();
        }
        void timer_Elapsed(object sender, EventArgs e)
        {
            drawSkeleton();
        }
        void timer_Show(object sender, EventArgs e)
        {
            var watch = Stopwatch.StartNew();
```

```
            gestureText.Text = gestureMeaning;
            PlaySlideShow(ctr);
            watch.Stop();
            var            computationTime            =
watch.ElapsedMilliseconds;
        }
        void timer_Data(object sender, EventArgs e)
        {
            cobaText.Text = sudutTangan.ToString();
            computationText.Text                       =
clusterPointCount.ToString();
        }
        private string sign = string.Empty;
        void element_Reset(object sender, EventArgs e)
        {
            this.sign = string.Empty;
            this.labelSign.Dispatcher.BeginInvoke(new
Action(() =>
            {
                this.labelSign.Content = this.sign;
            }));
        }
        void        element_Tick(object        sender,
System.Timers.ElapsedEventArgs e)
        {
            sign     =     sign     +     (sender     as
HandInterfaceElement).Number.ToString();
            this.labelSign.Dispatcher.Invoke(new
Action(() =>
            {
                this.labelSign.Content = this.sign;
            }));
        }
        private   void   Window_Closed(object   sender,
EventArgs e)
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
            Environment.Exit(0);
        }
        private   void   Button_Click(object   sender,
RoutedEventArgs e)
        {
```

```
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
            this.Close();
            Environment.Exit(0);
        }
        private  void  Window_Closing(object  sender,
System.ComponentModel.CancelEventArgs e)
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
            Environment.Exit(0);
        }
        private    void    checkHandLayer_Click(object
sender, RoutedEventArgs e)
        {
            this.ToggleLayers();
        }
        private  void  checkClusterLayer_Click(object
sender, RoutedEventArgs e)
        {
            this.ToggleLayers();
        }
        private    void    checkSkelLayer_Click(object
sender, RoutedEventArgs e)
        {
            System.Windows.Controls.CheckBox    cb    =
sender as System.Windows.Controls.CheckBox;
            if (cb.IsChecked == true)
            {
                kinectAdapter.StartTracking();
                checkJointName.IsEnabled = true;
            }
            if (cb.IsChecked == false)
            {
                kinectAdapter.StopTracking();
                checkJointName.IsEnabled = false;
            }
        }
        private    void    checkJointName_Click(object
sender, RoutedEventArgs e)
        {
```

```
            System.Windows.Controls.CheckBox    cb    =
sender as System.Windows.Controls.CheckBox;
            if (cb.IsChecked == true)
            {
                kinectAdapter.ShowNames = true;
            }
            if (cb.IsChecked == false)
            {
                kinectAdapter.ShowNames = false;
            }
        }
        private void ToggleLayers()
        {
            var layers = new List<IWpfLayer>();
            if
(this.checkHandLayer.IsChecked.GetValueOrDefault())
            {
                layers.Add(new
WpfHandLayer(this.handDataSource));
            }
            if
(this.checkClusterLayer.IsChecked.GetValueOrDefault())
            {
                layers.Add(new
WpfClusterLayer(this.clusterDataSource));
            }
            this.videoControl.Layers = layers;
        }
        void   MainWindow_NewDataAvailable(ImageSource
data)
        {
            this.videoControl.Dispatcher.Invoke(new
Action(() =>
            {

this.videoControl.ShowImageSource(data);
            }));
        }
        public void drawSkeleton()
        {
            canvasSkeleton.Children.Clear();
            TuioContact[] contacts;
            TuioContact contact;
            PointCollection points;
```

```
            List<TuioContact>     activeContacts     =
kinectAdapter.getContacts();
            for (int i = 0; i < kinect_points.Length;
i++)
            {
                points = new PointCollection();
                contacts = kinect_points[i];
                for (int j = 0; j < contacts.Length;
j++)
                {
                    contact = contacts[j];
                    if (contact.active)
                        points.Add(new
System.Windows.Point(contact.xpos * 640, contact.ypos
* 480));
                }
                drawLine(points);
            }
            for (int i = 0; i < activeContacts.Count;
i++)
            {
                contact = activeContacts[i];
            }
            points = new PointCollection();
            float handX = (1 - kinect.HandLeft.xpos) *
640;
            float handY = kinect.HandLeft.ypos * 480;
            points.Add(new  Point(handX  -  70,  handY  -
70));
            points.Add(new  Point(handX  +  70,  handY  -
70));
            points.Add(new  Point(handX  +  70,  handY  +
70));
            points.Add(new  Point(handX  -  70,  handY  +
70));
            points.Add(points[0]);
            drawLine(points);
            drawHand(fingerPointsDraw,       handDraw,
labelsDraw);
        }
        private  void  drawPoint(TuioContact  contact,
Brush brush)
        {
            Ellipse ellipse = new Ellipse();
            ellipse.Width = 12;
```

```
            ellipse.Height = 12;

            Canvas.SetLeft(ellipse, contact.xpos * 640
- 6);
            Canvas.SetTop(ellipse, contact.ypos * 480
- 6);
            ellipse.Stroke = brush;
            ellipse.StrokeThickness = 12;
            canvasSkeleton.Children.Add(ellipse);
        }

        private void drawLine(PointCollection points)
        {
            Polyline polyline = new Polyline();
            polyline.Points = points;
            polyline.Stroke = redBrush;
            polyline.StrokeThickness = 5;
            canvasSkeleton.Children.Add(polyline);
        }
        private    void    drawHand(List<TuioContact>
fingerPoints,   TuioContact   hand,   List<TextBlock>
labels)
        {
            for (int i = 0; i < fingerPoints.Count;
i++)
            {
                TuioContact finger = fingerPoints[i];
                if (finger.active)
                {
                    PointCollection   points   =   new
PointCollection();
                    points.Add(new
System.Windows.Point(handPalm.X, handPalm.Y));
                    points.Add(new
System.Windows.Point((finger.xpos) * 640, finger.ypos
* 480));
                    Canvas.SetLeft(labels[i],
(finger.xpos * 640));
                    Canvas.SetTop(labels[i],
finger.ypos * 480 - 15);
canvasSkeleton.Children.Add(labels[i]);
                    drawLine(points);
                    drawPoint(finger, greenBrush);
                }
            }
```

```
        }
        void
handDataSource_NewDataAvailable(HandCollection data)
        {
            if (!handType)
            {
                this.Thumb = kinectAdapter.ThumbLeft;
                this.IndexFinger                    =
kinectAdapter.IndexFingerLeft;
                this.MiddleFinger                   =
kinectAdapter.MiddleFingerLeft;
                this.RingFinger                     =
kinectAdapter.RingFingerLeft;
                this.Pinky = kinectAdapter.PinkyLeft;
                this.Hand = kinectAdapter.HandLeft;
                this.Wrist = kinectAdapter.WristLeft;
                AngleScalingFactor = -180.0 / Math.PI;
            }
            else
            {
                this.Thumb = kinectAdapter.ThumbRight;
                this.IndexFinger                    =
kinectAdapter.IndexFingerRight;
                this.MiddleFinger                   =
kinectAdapter.MiddleFingerRight;
                this.RingFinger                     =
kinectAdapter.RingFingerRight;
                this.Pinky = kinectAdapter.PinkyRight;
                this.Hand = kinectAdapter.HandRight;
                this.Wrist = kinectAdapter.WristRight;
                AngleScalingFactor = 180.0 / Math.PI;
            }
            if (gestureChooseThread == false)
            {
                this.numberGestureClass(data);
            }
            else
            {
                this.letterGestureClass(data);
            }
            if (!data.IsEmpty)
            {
                this.handPalm.X = data.Hands[0].PalmX;
                this.handPalm.Y = data.Hands[0].PalmY;
```

```
                double          bufferSudut          =
calculateVector(data);
                if(!handType)
                {
                    if(bufferSudut < 0)
                    {
                        this.sudutTangan             =
(bufferSudut + 180) * -1;
                    }
                    else
                    {
                        this.sudutTangan     =     180-
bufferSudut;
                    }
                }
                else
                {
                    this.sudutTangan = bufferSudut;
                }
                this.clusterPointCount               =
clusterDataSource.CurrentValue.Clusters.First().PointC
ount;
            }
            this.interfaceElement.Update(data);
        }
        private         void         numberGestureClass
(HandCollection data)
        {
            if (!data.IsEmpty)
            {
                if  (data.Hands.First().FingerCount  ==
0 && data.HandsDetected)
                {
                    gestureMeaning = "Nol";
                    ctr = 0;
                }
                else                                  if
(data.Hands.First().FingerCount == 1)
                {
                    if (IndexFinger.active)
                    {
                        gestureMeaning = "Satu";
                        ctr = 1;
                    }
                }
```

```
                else                                    if
(data.Hands.First().FingerCount == 2)
                {
                    if      (IndexFinger.active      &&
MiddleFinger.active)
                    {
                        gestureMeaning = "Dua";
                        ctr = 2;
                    }
                }
                else                                    if
(data.Hands.First().FingerCount == 3)
                {
                    if      (IndexFinger.active      &&
MiddleFinger.active && Thumb.active)
                    {
                        gestureMeaning = "Tiga";
                        ctr = 3;
                    }
                    else  if  (  IndexFinger.active  &&
MiddleFinger.active && RingFinger.active)
                    {
                        gestureMeaning = "Enam";
                        ctr = 6;
                    }
                    else  if  (IndexFinger.active  &&
MiddleFinger.active && Pinky.active)
                    {
                        gestureMeaning = "Tujuh";
                        ctr = 7;
                    }
                    else  if  (IndexFinger.active  &&
RingFinger.active && Pinky.active)
                    {
                        gestureMeaning = "Delapan";
                        ctr = 8;
                    }
                    else  if  (RingFinger.active  &&
MiddleFinger.active && Pinky.active)
                    {
                        gestureMeaning = "Sembilan";
                        ctr = 9;
                    }
                }
```

```
                else                                    if
(data.Hands.First().FingerCount == 4)
                {
                    if      (IndexFinger.active      &&
MiddleFinger.active     &&     RingFinger.active     &&
Pinky.active)
                    {
                        gestureMeaning = "Empat";
                        ctr = 4;
                    }
                }
                else                                    if
(data.Hands.First().FingerCount == 5)
                {
                    gestureMeaning = "Lima";
                    ctr = 5;
                }
            }
        }
        private void letterGestureClass(HandCollection
data)
        {
            if (!data.IsEmpty)
            {
                if  (data.Hands.First().FingerCount  ==
0 && data.HandsDetected)
                {
                    if  (clusterPointCount  >=  155  &&
clusterPointCount < 190)
                    {
                        gestureMeaning = "E";
                        ctr = 14;
                    }
                    else if (clusterPointCount >= 140
&& clusterPointCount < 155)
                    {
                        gestureMeaning = "A";
                        ctr = 10;
                    }
                    else if (clusterPointCount >= 125
&& clusterPointCount < 140)
                    {
                        gestureMeaning = "0";
                        ctr = 23;
                    }
```

```
                    else if (clusterPointCount >= 110
&& clusterPointCount < 125)
                    {
                        gestureMeaning = "S";
                        ctr = 27;
                    }
                    else if (clusterPointCount >= 90
&& clusterPointCount < 110)
                    {
                        gestureMeaning = "T";
                        ctr = 28;
                    }
                    else if (clusterPointCount < 90)
                    {
                        gestureMeaning = "M / N";
                        ctr = 21;
                    }
                }
                else                                    if
(data.Hands.First().FingerCount == 1)
                {
                    if (IndexFinger.active)
                    {
                        if  (sudutTangan  >=  110  &&
sudutTangan <= 130)
                        {
                            gestureMeaning = "X";
                            ctr = 32;
                        }
                        else if (sudutTangan >= 85 &&
sudutTangan < 110)
                        {
                            if  (clusterPointCount  >=
105 && clusterPointCount < 125)
                            {
                                gestureMeaning = "D";
                                ctr = 13;
                            }
                            else if (clusterPointCount
>= 125 && clusterPointCount < 143)
                            {
                                gestureMeaning = "R";
                                ctr = 26;
                            }
```

```
                            else if (clusterPointCount
>= 143 && clusterPointCount < 160)
                            {
                                gestureMeaning = "U";
                                ctr = 29;
                            }
                        }
                    }
                    else if (Thumb.active)
                    {
                        if  (sudutTangan  >=  -180  &&
sudutTangan <= -169)
                        {
                            gestureMeaning = "C";
                            ctr = 12;
                        }
                        else if (sudutTangan >= 155 &&
sudutTangan <= 170)
                        {
                            gestureMeaning = "G";
                            ctr = 16;
                        }
                    }
                    else if (Pinky.active)
                    {
                        gestureMeaning = "I";
                        ctr = 18;
                    }
                }
                else                                    if
(data.Hands.First().FingerCount == 2)
                {
                    if      (IndexFinger.active      &&
Thumb.active)
                    {
                        if   (sudutTangan   >=   125   &&
sudutTangan <= 140)
                        {
                            gestureMeaning = "L";
                            ctr = 20;
                        }
                        else if (sudutTangan >= 105 &&
sudutTangan < 125)
                        {
                            gestureMeaning = "K";
```

```
                        ctr = 19;
                    }
                    else if (sudutTangan >= -180
&& sudutTangan <= -160)
                    {
                        gestureMeaning = "P";
                        ctr = 24;
                    }
                }
                else if (IndexFinger.active &&
MiddleFinger.active)
                {
                    if (sudutTangan >= 155 &&
sudutTangan <= 177)
                    {
                        gestureMeaning = "H";
                        ctr = 17;
                    }
                    else if (sudutTangan >= 75 &&
sudutTangan <= 115)
                    {
                        gestureMeaning = "V";
                        ctr = 30;
                    }
                }
                else    if    (Pinky.active    &&
Thumb.active)
                {
                    if (sudutTangan >= -110 &&
sudutTangan <= -80)
                    {
                        gestureMeaning = "Q";
                        ctr = 25;
                    }
                    else if (sudutTangan >= 60 &&
sudutTangan <= 115)
                    {
                        gestureMeaning = "Y";
                        ctr = 33;
                    }
                }
            }
            else                                if
(data.Hands.First().FingerCount == 3)
            {
```

```
                    if      (IndexFinger.active      &&
MiddleFinger.active && RingFinger.active)
                    {
                        gestureMeaning = "W";
                        ctr = 31;
                    }
                    else  if  (MiddleFinger.active  &&
RingFinger.active && Pinky.active)
                    {
                        gestureMeaning = "F";
                        ctr = 15;
                    }
                }
                else                                    if
(data.Hands.First().FingerCount == 4)
                {
                    if      (IndexFinger.active      &&
MiddleFinger.active     &&     RingFinger.active     &&
Pinky.active)
                    {
                        gestureMeaning = "B";
                        ctr = 11;
                    }
                }
            }
        }
        private void PlaySlideShow(int ctr)
        {
            BitmapImage image = new BitmapImage();
            image.BeginInit();
            string  filename  =  "Images/Sign"  +  ctr  +
".jpg";
            image.UriSource   =   new   Uri(filename,
UriKind.Relative);
            image.EndInit();
            signImage.Source = image;
            signImage.Stretch = Stretch.Uniform;
        }

        private void angkaRadio_Checked(object sender,
RoutedEventArgs e)
        {
            gestureChooseThread = false;
        }
```

```
        private void hurufRadio_Checked(object sender,
RoutedEventArgs e)
        {
            gestureChooseThread = true;
        }

        private void leftRadio_Checked(object sender,
RoutedEventArgs e)
        {
            if (handChooseThread)
            {
                this.handType = false;
                this.fingerPointsDraw = fingersLeft;
                this.handDraw                          =
kinectAdapter.HandLeft;
                this.labelsDraw = labelsLeft;
            }
        }
        private void rightRadio_Checked(object sender,
RoutedEventArgs e)
        {
            if (handChooseThread)
            {
                this.handType = true;
                this.fingerPointsDraw = fingersRight;
                this.handDraw                          =
kinectAdapter.HandRight;
                this.labelsDraw = labelsRight;
            }
        }
        private double calculateVector(HandCollection
data)
        {
            //double angle;
            double vectorX = 0;
            double vectorY = 0;
            double fingerVectorX = 0;
            double fingerVectorY = 0;
            IList<FingerPoint>     fingerPoints     =
data.Hands[0].FingerPoints;
            for (int i = 0; i < fingerPoints.Count;
i++)
            {
                FingerPoint finger = fingerPoints[i];
```

```
                CCT.NUI.Core.Point      location      =
finger.Location;
                fingerVectorX  =  ((finger.X  *  640)  -
(data.Hands[0].PalmX * 640));
                fingerVectorY  =  ((finger.Y  *  480)  -
(data.Hands[0].PalmY * 480));
                vectorX = vectorX + fingerVectorX;
                vectorY = vectorY + fingerVectorY;
            }
            double  vectorHand  =  Math.Sqrt((vectorX  *
vectorX) + (vectorY * vectorY));
            double          vectorHandRadian          =
Math.Atan2(vectorY, vectorX);

            double          vectorHandDegree          =
(vectorHandRadian * (180/Math.PI)) * -1;
            return vectorHandDegree;
        }
        private  void  resetButton_Click(object  sender,
RoutedEventArgs e)
        {
            this.interfaceElement.ResetText();
        }
    }
}
```

BIOGRAFI PENULIS

Hendra Irawan dilahirkan di Kota Sumenep, Madura pada tanggal 08 Januari 1994. Memulai Sekolah Dasar di SDN Pangarangan III tahun 1999 hingga 2005. Kemudian Penulis melanjutkan bersekolah di SMPN 1 Sumenep hingga tahun 2008. Jenjang selanjutnya, Penulis bersekolah di SMA Negeri 3 Pamekasan hingga tahun 2011. Sejak tahun 2011 penulis melanjutkan pendidikan di Jurusan Teknik Fisika Fakultas Teknologi Industri ITS Surabaya. Selama terdaftar sebagai mahasiswa, karena ketertarikannya pada dunia robottika penulis sempat menjadi anggota Tim Robot ITS divisi Beroda sebagai Electrical Engineer. Kemudian karena ketertarikan penulis terhadap dunia Otomotif, penulis juga sempat menjadi anggota Tim Sapu Angin ITS untuk berkompetisi pada ajang Indonesia Energy Marathon Challenge, Shell Eco Marathon Asia di Filipina, dan SAE Student Formula Japan sebagai Electric Instrument Engineer. Pada rentang waktu Januari – Februari 2015 melaksanakan kerja praktek di PT. Dirgantara Indonesia dengan tema penelitian mengenai *"Studi dan Analisa Sistem Kontrol Indikator Bahan Bakar di Pesawat Fixed Wing C212-400"*. Penulis juga pernah menerbitkan poster ilmiah untuk diikutkan pad Pekan Ilmiah Mahasiswa Nasional (PIMNAS) 26 pada tahun 2013 dengan judul "*U.S.S System (Ultrasonic Security System), Sistem keamanan ruang berbasis ultrasonik yang efektif dan efisien".* Penulis dapat dihubungi melalui email neuhendra@gmail.com.